Introduction to Deep Learning for Geoscience

Ronan Fablet (ronan.fablet@ilt-atlantique.fr)

Course on “Data Science for Geosciences”, DSG’2020, Toulouse, January 2020

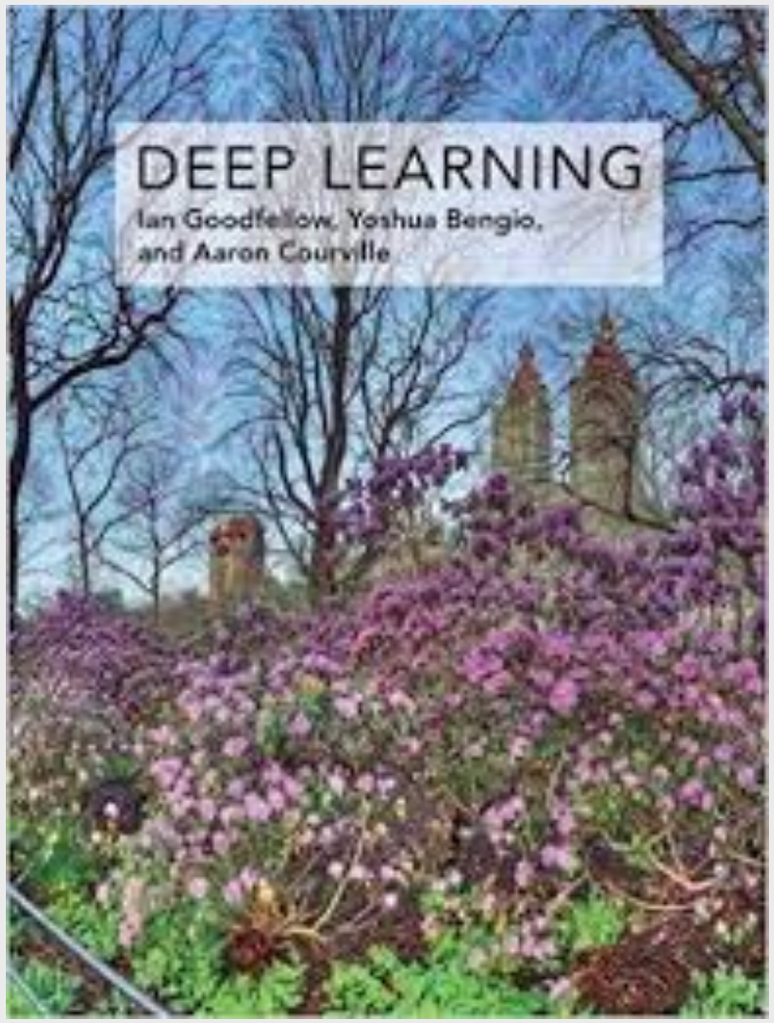
This course focuses on “**an introduction to deep learning**", not just "**how to use**” deep learning in practice.

Resources

- **Book: Deep Learning**

Goodfellow, Bengio, Courville, MIT Press

Online version http://www.deeplearningbook.org/



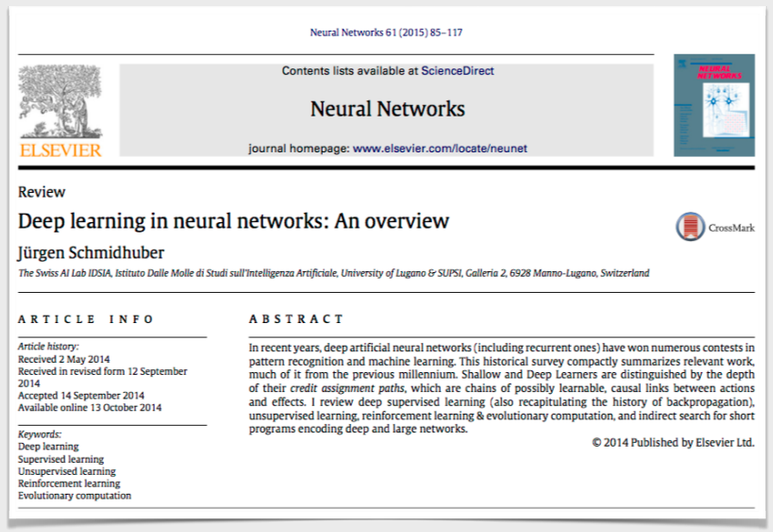
- **Online course by Andrew Ng (Stanford/Baidu)**

  Youtube: link

  Online course on Coursera: link

- **Review paper: Deep learning in neural networks by J. Schmidhuber**

  pdf: link


Neural Networks 61 (2015) 85–117

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Review

Deep learning in neural networks: An overview

Jürgen Schmidhuber

The Swiss AI Lab IDSIA, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, University of Lugano & SUPSI, Galleria 2, 6928 Manno-Lugano, Switzerland

ARTICLE INFO

Article history:
Received 2 May 2014
Received in revised form 12 September 2014
Accepted 14 September 2014
Available online 13 October 2014

Keywords:
Deep learning
Supervised learning
Unsupervised learning
Reinforcement learning
Evolutionary computation

ABSTRACT

In recent years, deep artificial neural networks (including recurrent ones) have won numerous contests in pattern recognition and machine learning. This historical survey compactly summarizes relevant work, much of it from the previous millennium. Shallow and Deep Learners are distinguished by the depth of their *credit assignment paths*, which are chains of possibly learnable, causal links between actions and effects. I review deep supervised learning (also recapitulating the history of backpropagation), unsupervised learning, reinforcement learning & evolutionary computation, and indirect search for short programs encoding deep and large networks.

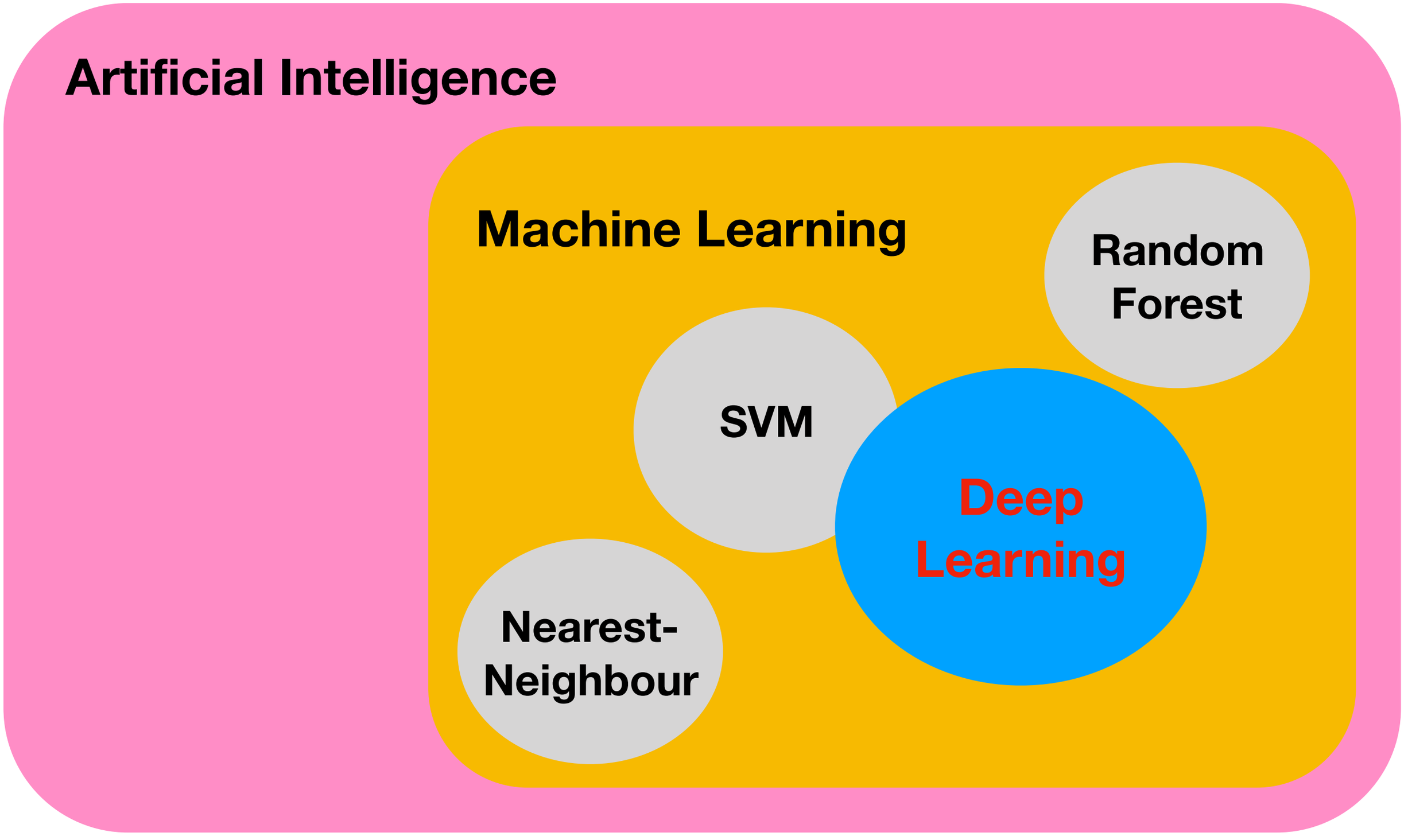
© 2014 Published by Elsevier Ltd.

Roadmap

- What is deep learning ?
- What does (machine) learning mean ?
- What are Neural Networks ?
- What are the key components of DL models ?
- How to implement DL models using Keras ?
- A (very) short tour of the NN zoo

What is Deep Learning?

What is Deep Learning?



Deep learning has rapidly become the state-of-art-framework for a wide range of applications in computer vision, natural language processing, signal processing…

From Image to Text

(image captioning)

A woman is throwing a **frisbee** in a park.

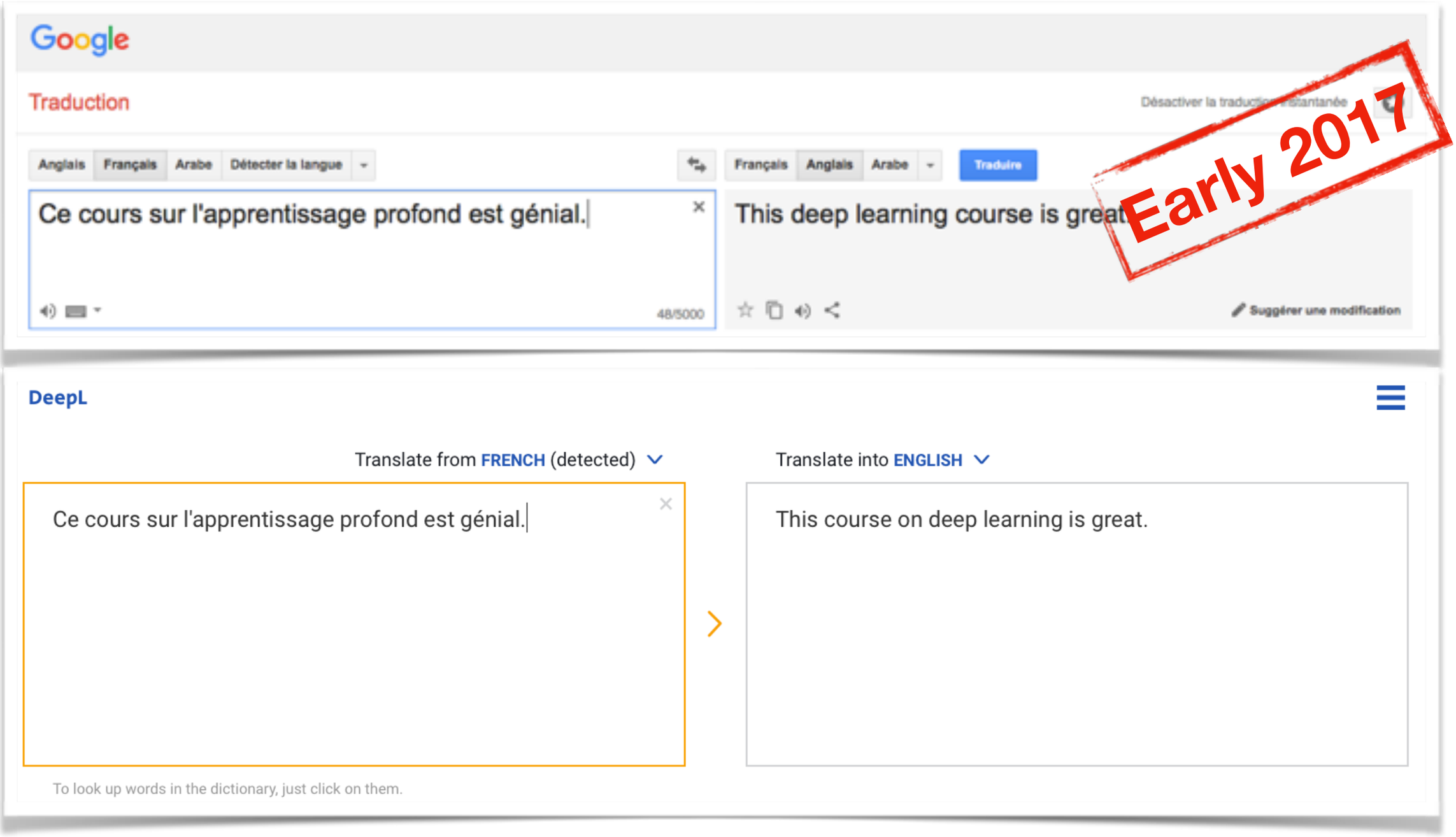
A **dog** is standing on a hardwood floor.

A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

Vinyals et al., ICML 2014

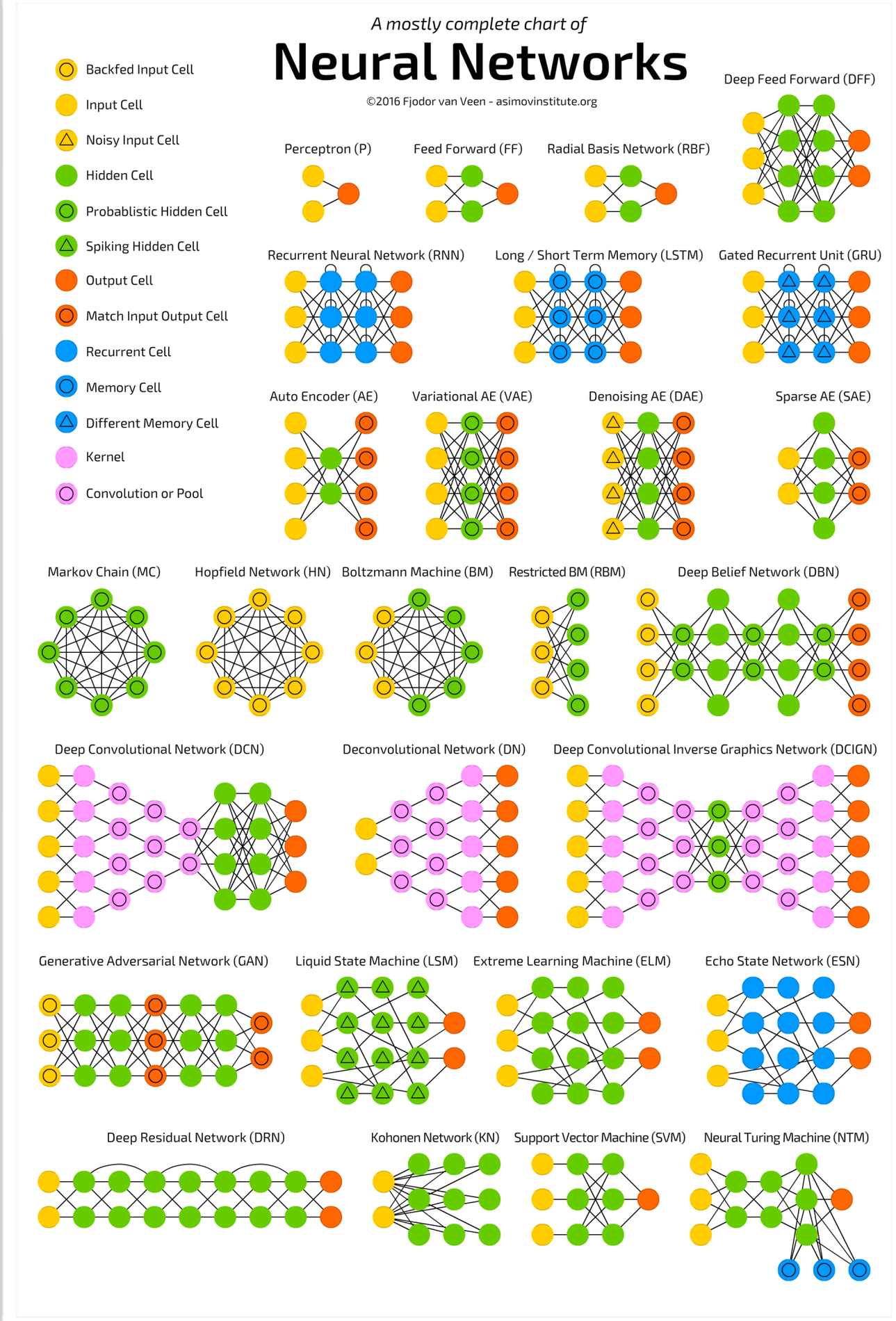
Automatic Translation



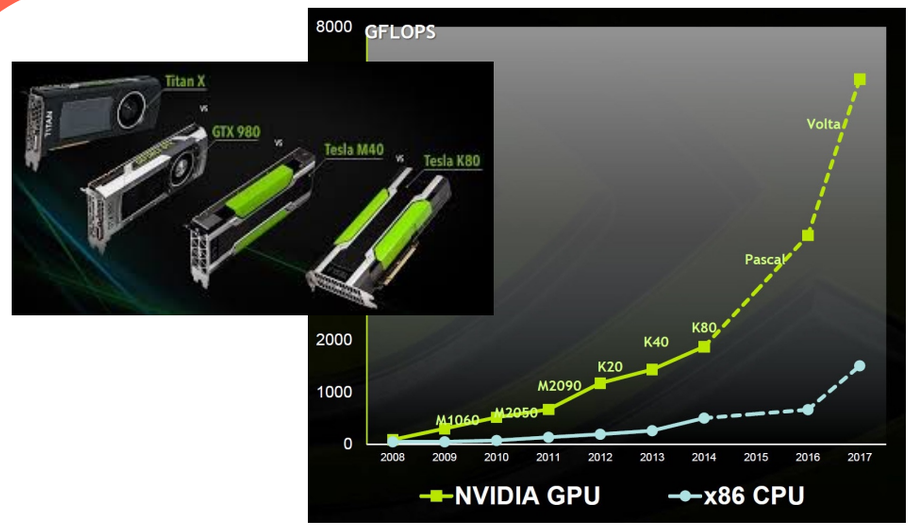
and many more…….

Deep learning relies on neural networks, which are not new…

Artificial neural networks date back to the 60's and were popular in the 80's.



Key reasons for DL emergence



High-performance computing (GPU)

Large annotated dataset (> 1M)

+

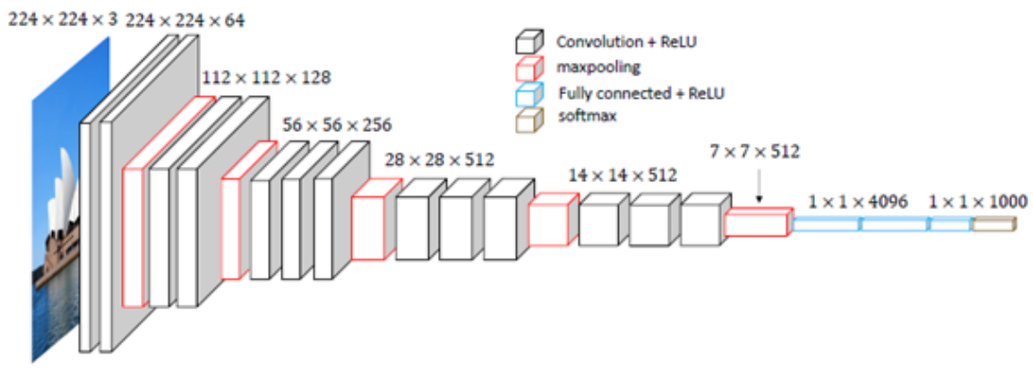
TensorFlow

Keras

Caffe

PYTORCH

Efficient & easy-to-use libraries



End-to-end learning

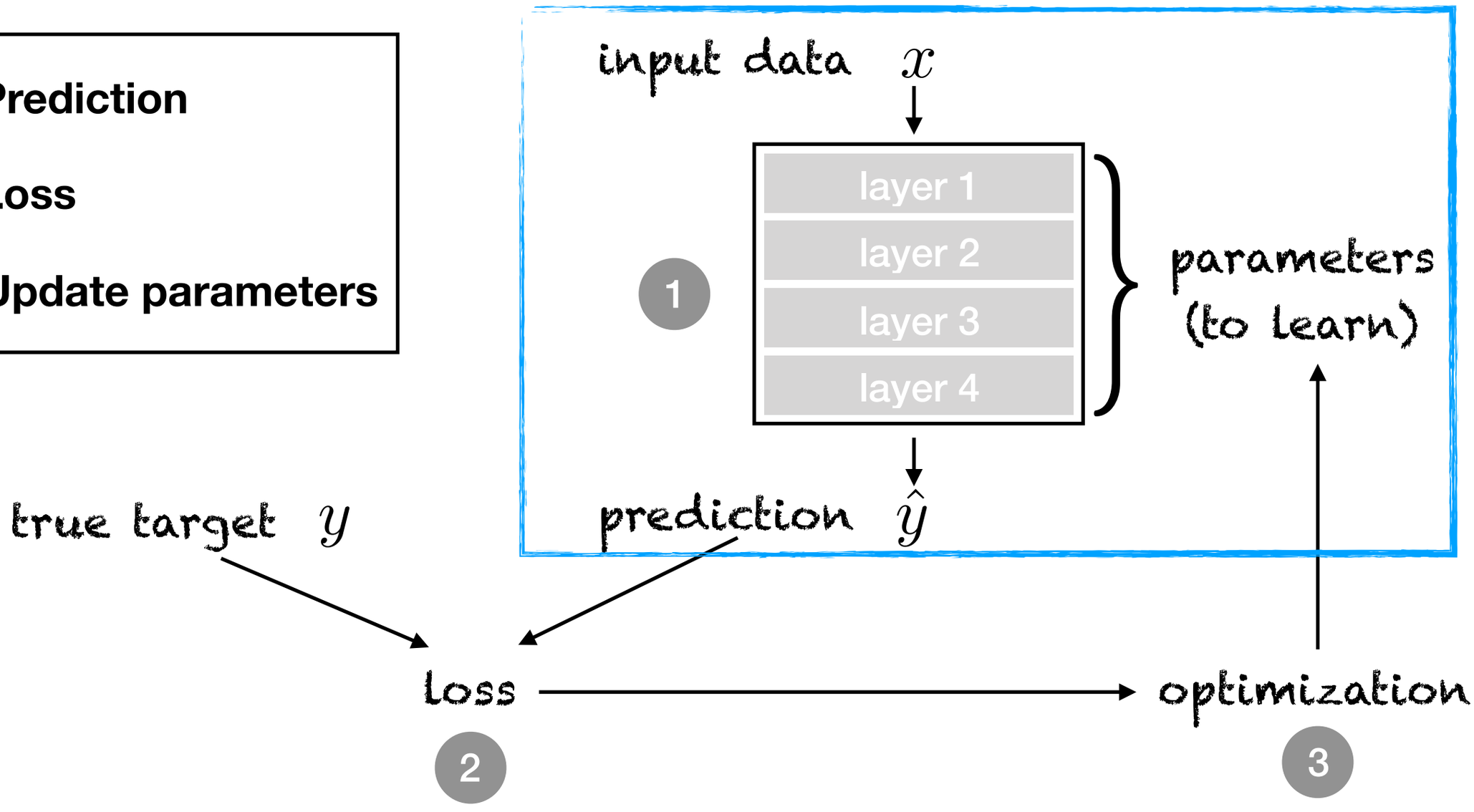
WHAT DOES (MACHINE) LEARNING MEAN ?

Machine/Deep Learning

Definition (wikipedia): Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

Machine learning

1. **Prediction**
2. **Loss**
3. **Update parameters**



Machine/Deep Learning

Mathematical formulation: Learning comes to minimising some loss function given w.r.t. model parameters and training data

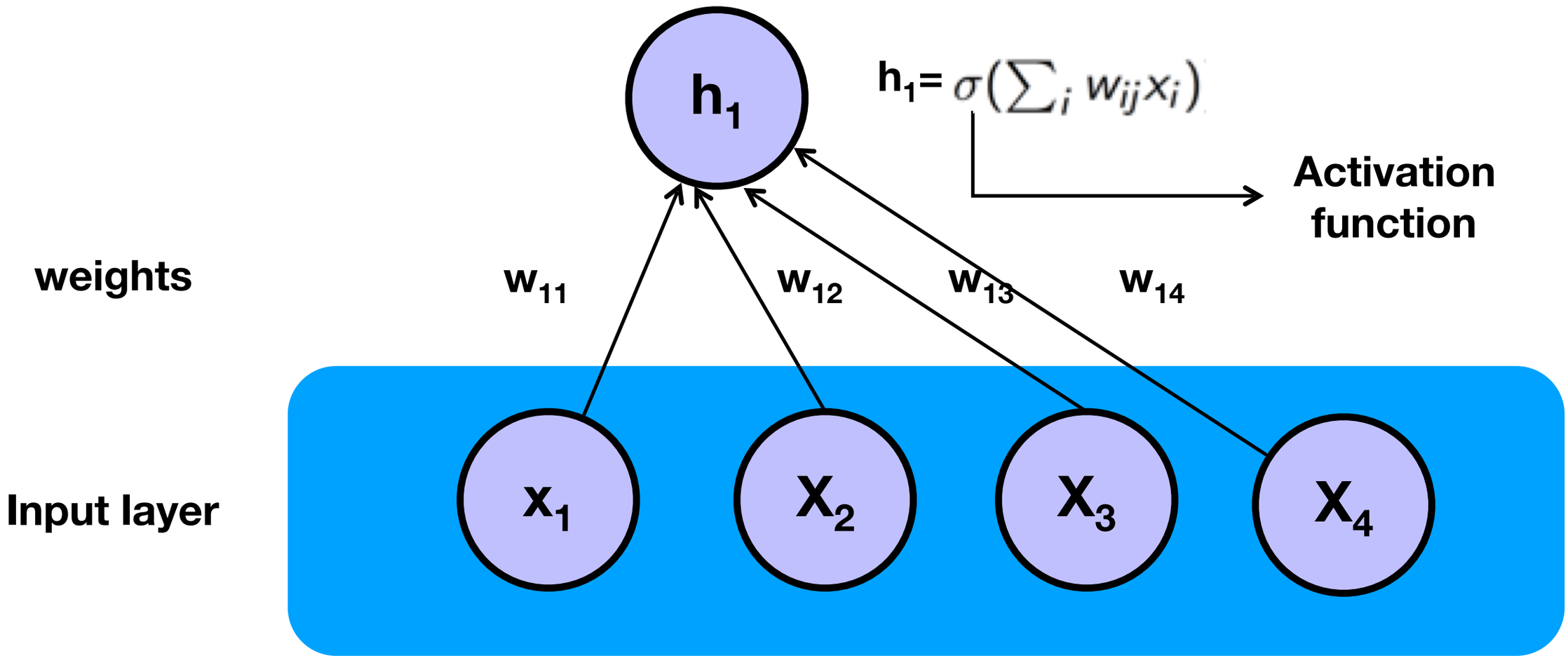
$$\widehat{\theta} = \arg\min_{\theta} \mathcal{L}\left(\{x_i, y_i\}_{i \in \{1,..,N\}}; f_\theta\right)$$

Key questions:

- Which parameterisation for model f ?
- Which loss function ?

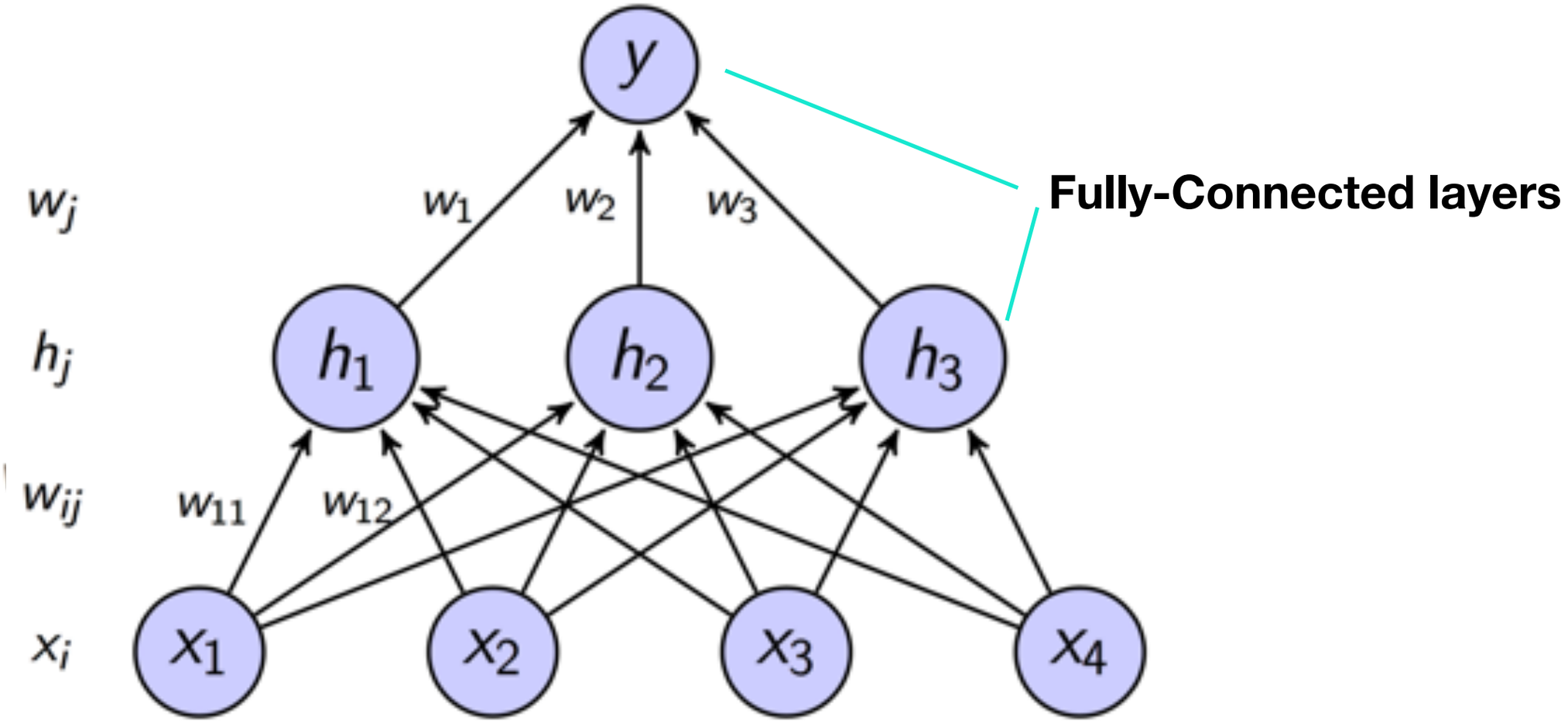
NEURAL NETWORKS: KEY PRINCIPLES & BUILDING BLOCKS

The artificial neuron



$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

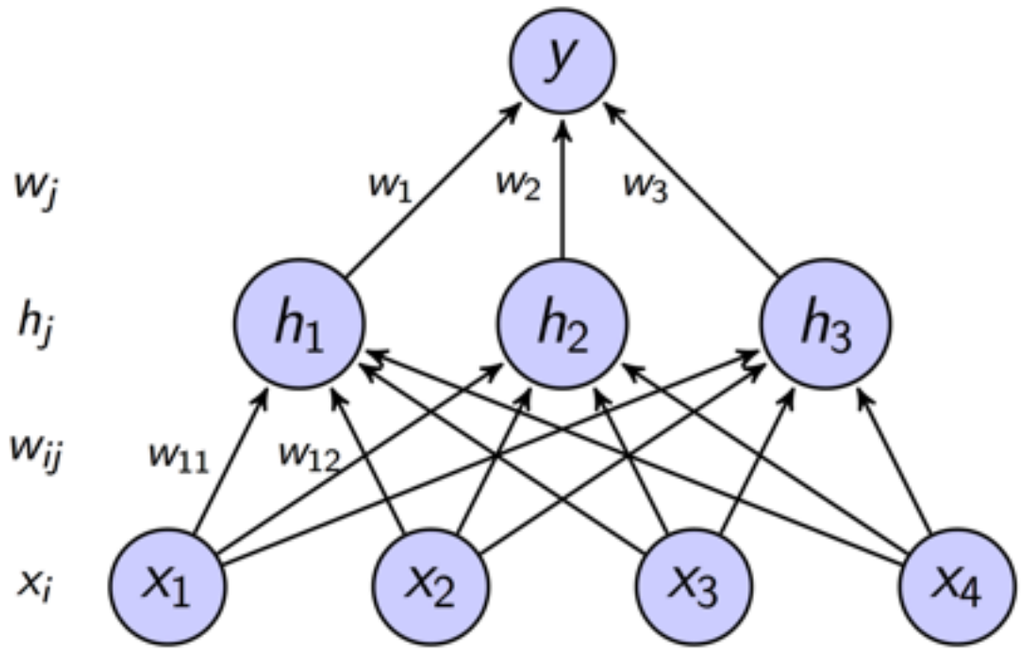
Multilayer Perceptron (MLP)



$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

Multilayer Perceptron (MLP)

- Neuron: basic unit, inspired from neuroscience
- Feedforward: no feedback connection
- Network: composition of functions
- Parametric model: $y=f(x_1, x_2, x_3, x_4, \boldsymbol{\theta})$
- Training: estimation of model parameter to minimise some loss function



Let’s play with MLPs using tensorflow playground

http://playground.tensorflow.org/

Neural networks: composition idea

- **Approximation through the composition of (simple) elementary functions:**

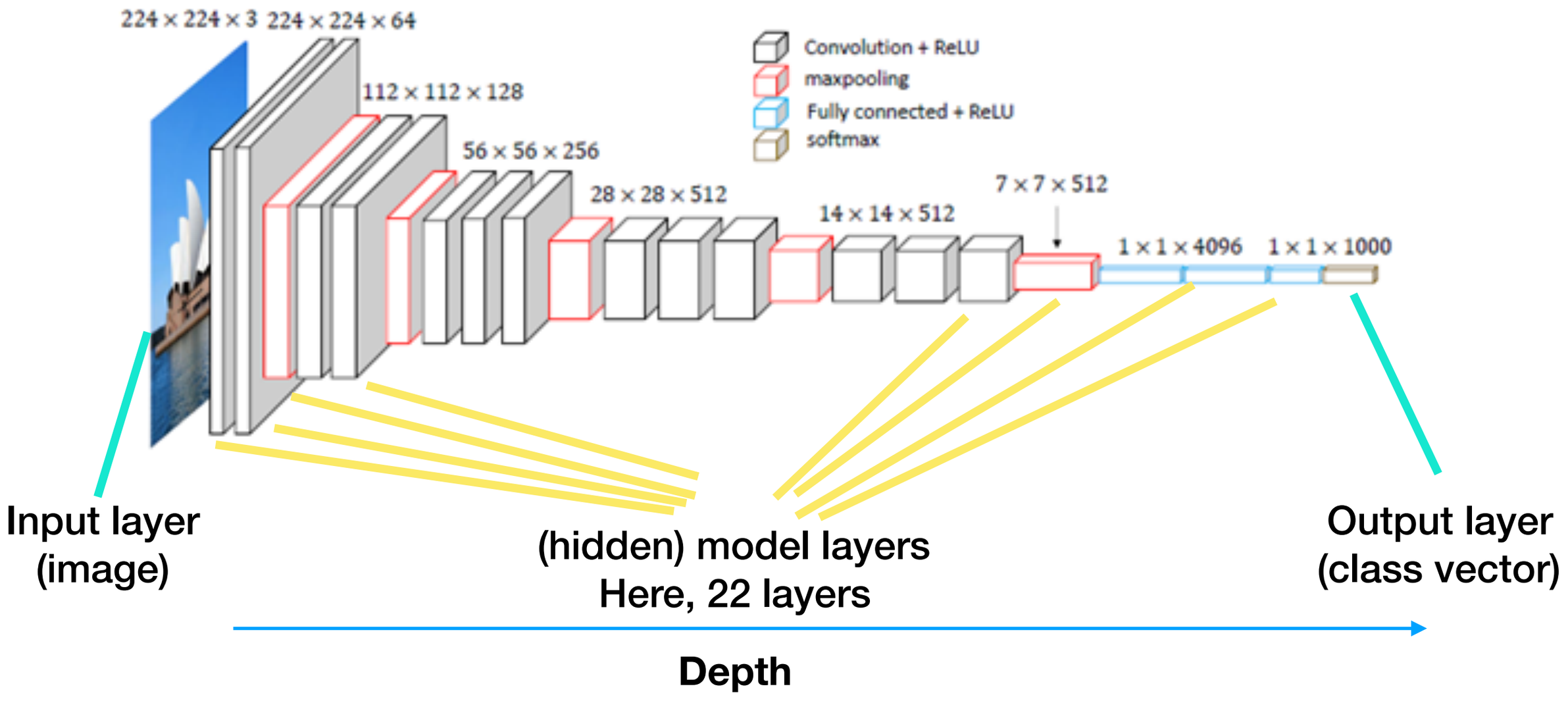
$$f_\theta(x) = f_{\theta_N} \circ \ldots \circ f_{\theta_2} \circ f_{\theta_1}(x)$$

- **Key features:**
  - Any continuous function can be approximated as the composition of elementary functions
  - Analytical/exact computation of the derivative of f with respect to parameters and input variables
  - Direct exploitation of gradient-based optimisation schemes

What are deep learning models ?

Basics of DL models

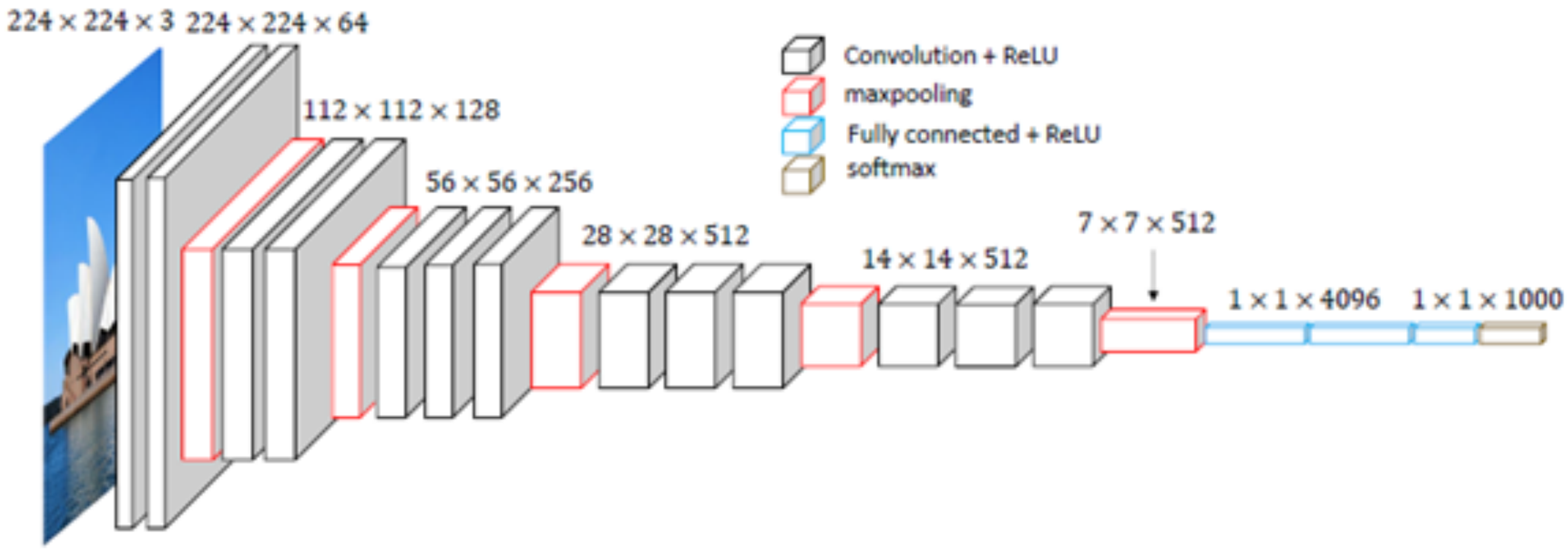
DL models are (in general) feedforward models. VGG16 as an illustration



The more layers, the deeper…… Some models may have up to several hundreds to thousands of layers.

Basics of DL models

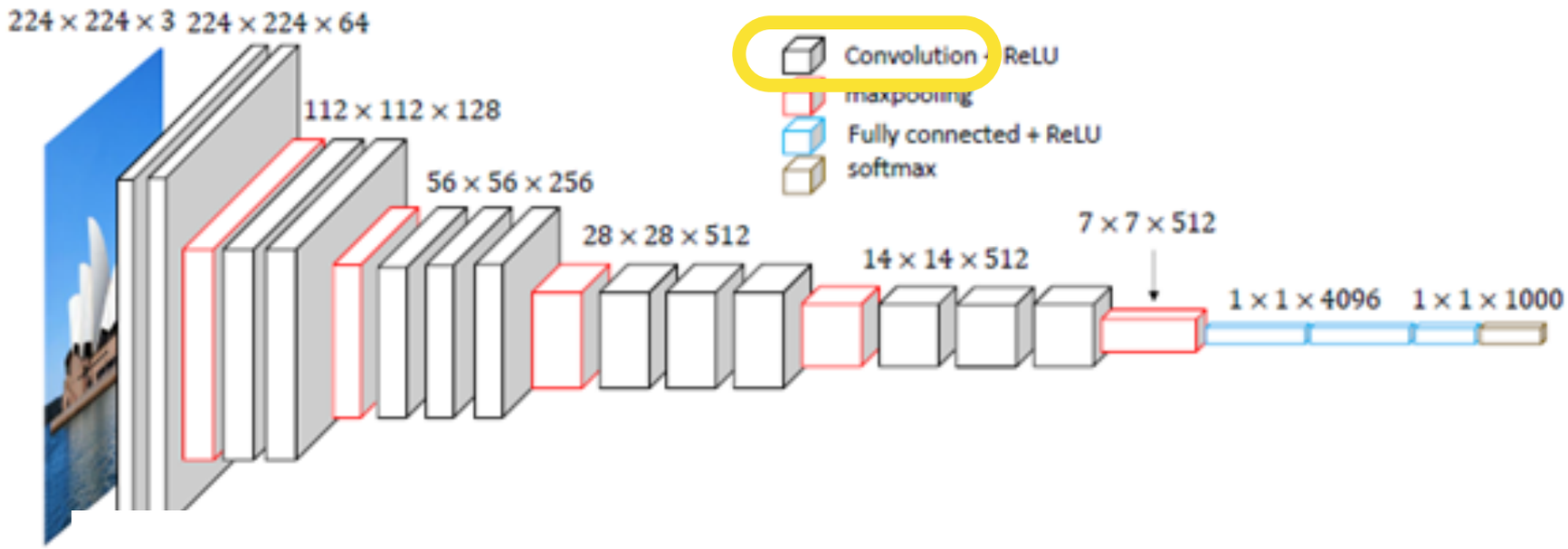
DL models are (in general) feedforward models. VGG16 as an illustration



Elementary components

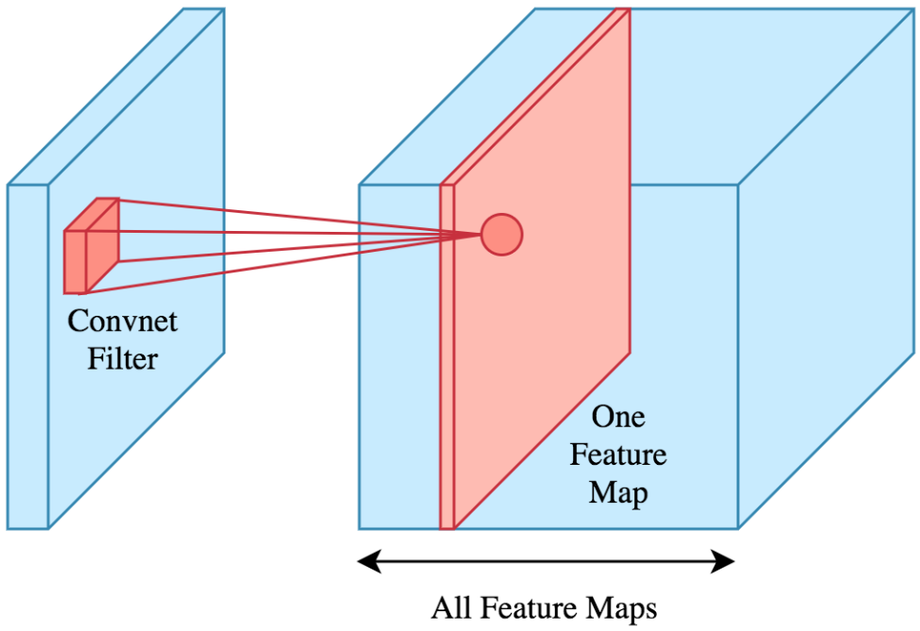
- Convolution layers
- Activation layers
- Pooling layers
- Dense layers

Basics of DL models



Elementary components

- Convolution layers
- Activation layers
- Pooling layers
- Dense layers

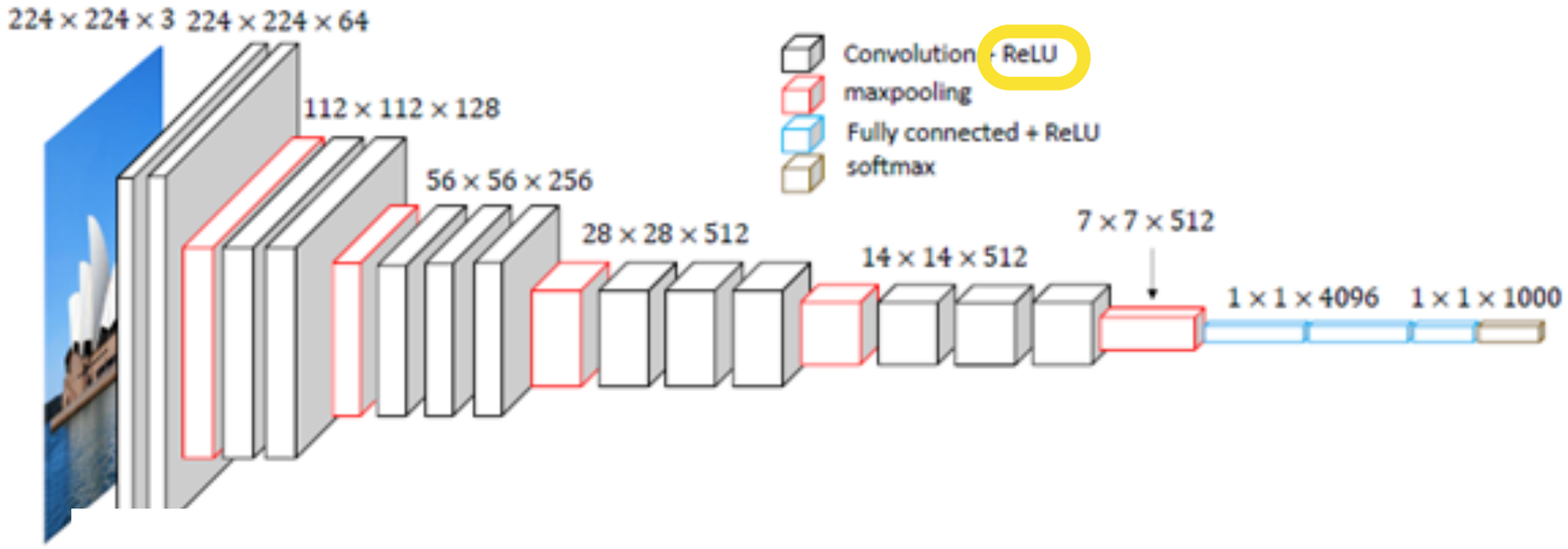


Number of parameters:
Filter size x Number of filters

e.g. 3x3xKxNfilt

Independent on the sizes of the input and output layer

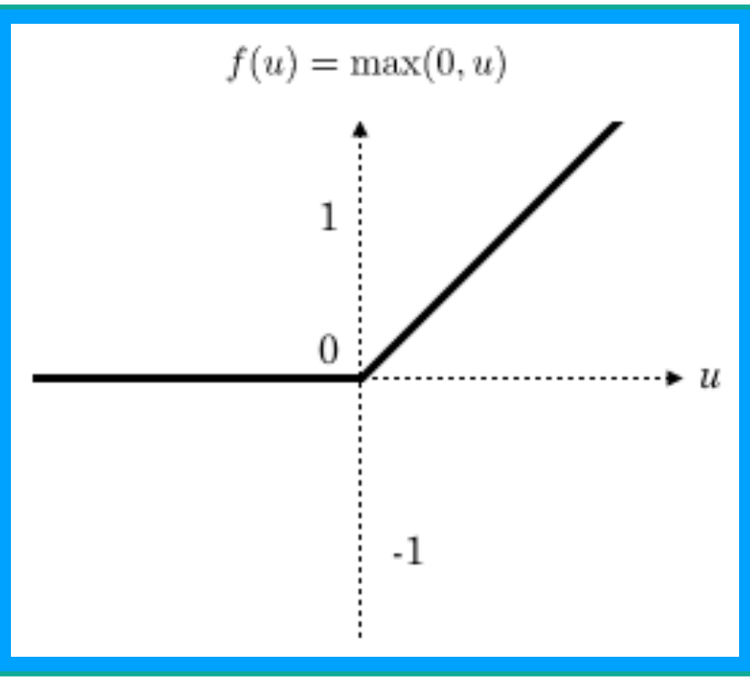
Basics of DL models

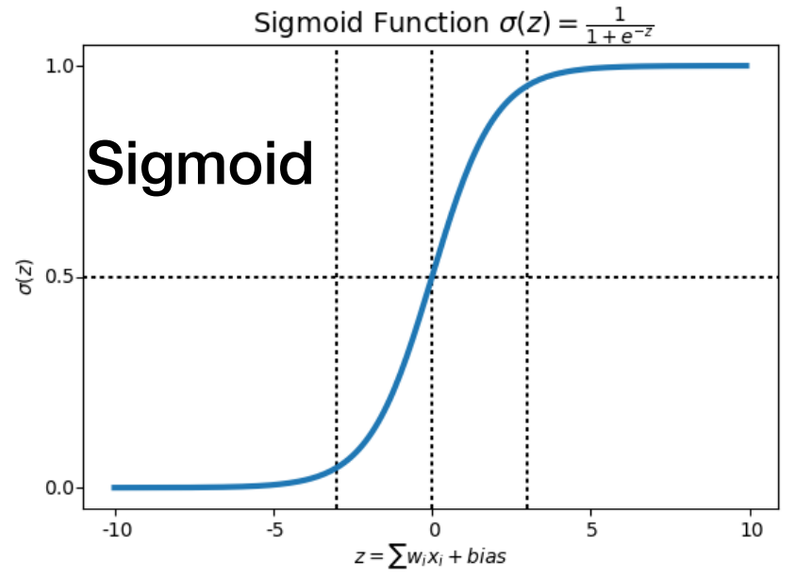


Elementary components

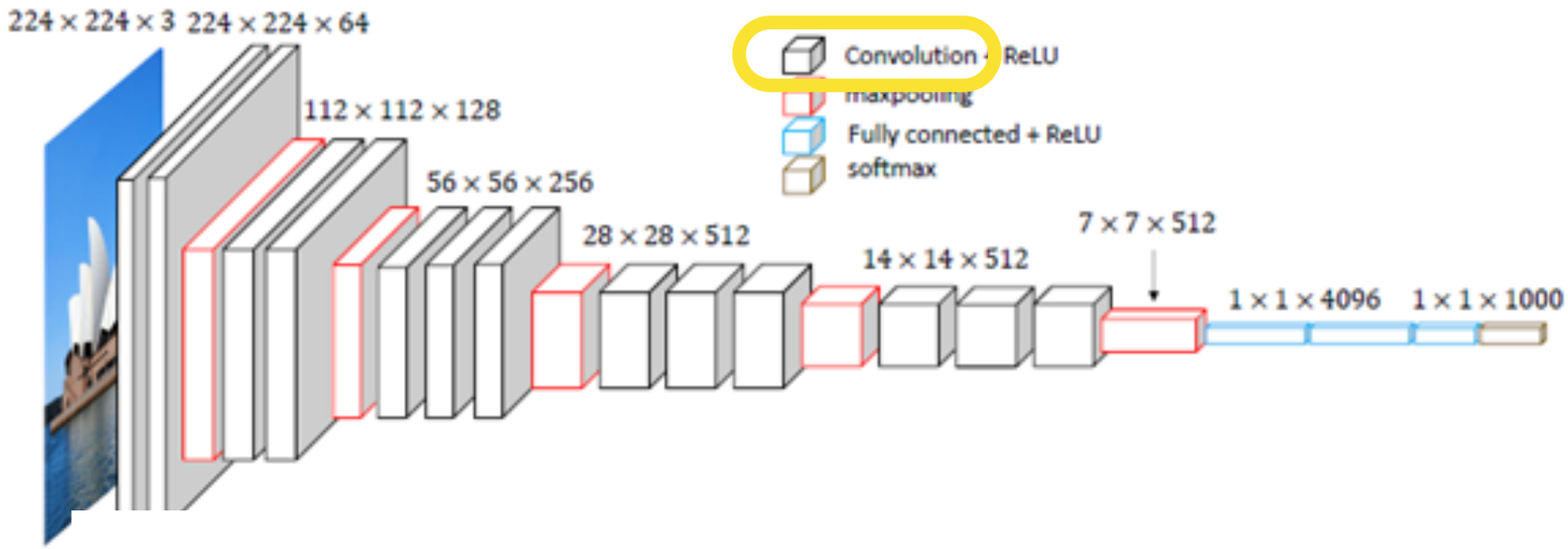
- Convolution layers
- Activation layers
- Pooling layers
- Dense layers

ReLU (Rectified Linear Unit)

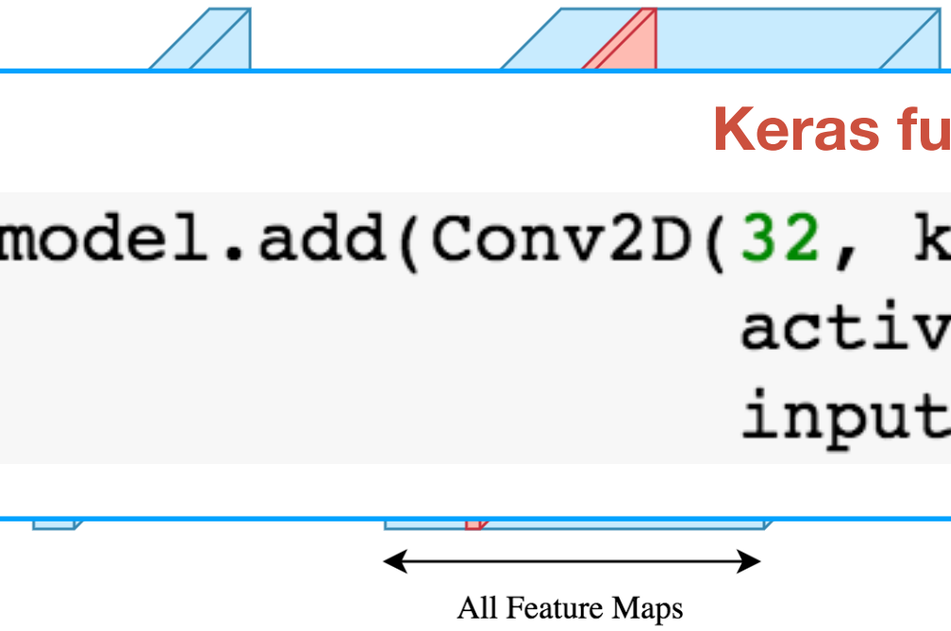




Basics of DL models



Elementary components	Convolution layers	Activation layers	Pooling layers	Dense layers



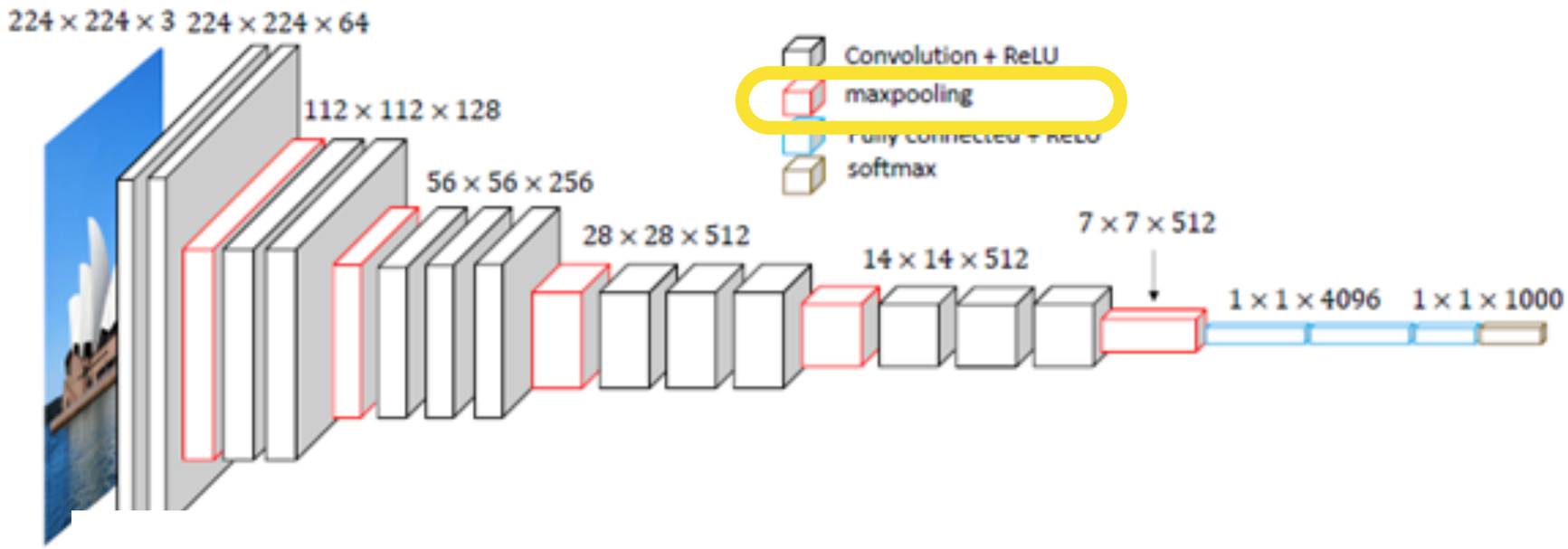
Number of parameters:

Keras function

```
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
```

put

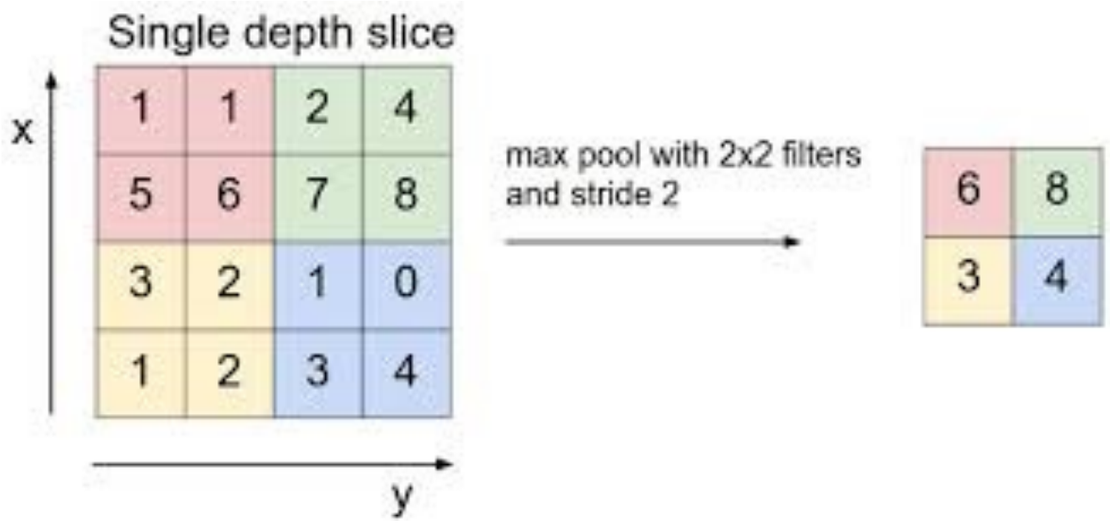
Basics of DL models



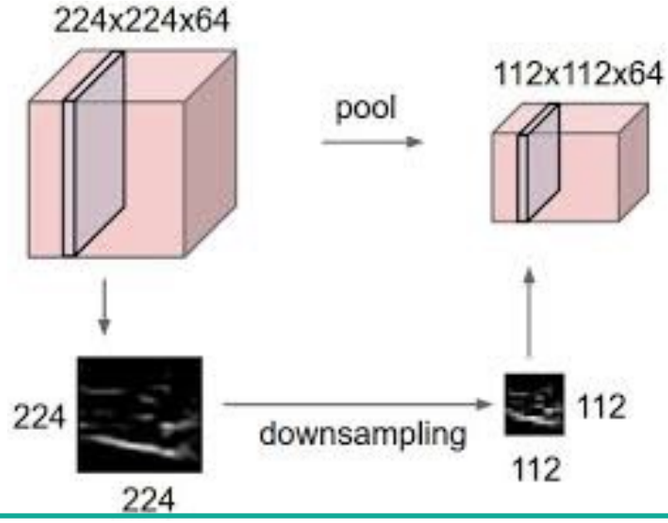
Elementary components

Convolution layers | Activation layers | **Pooling layers** | Dense layers

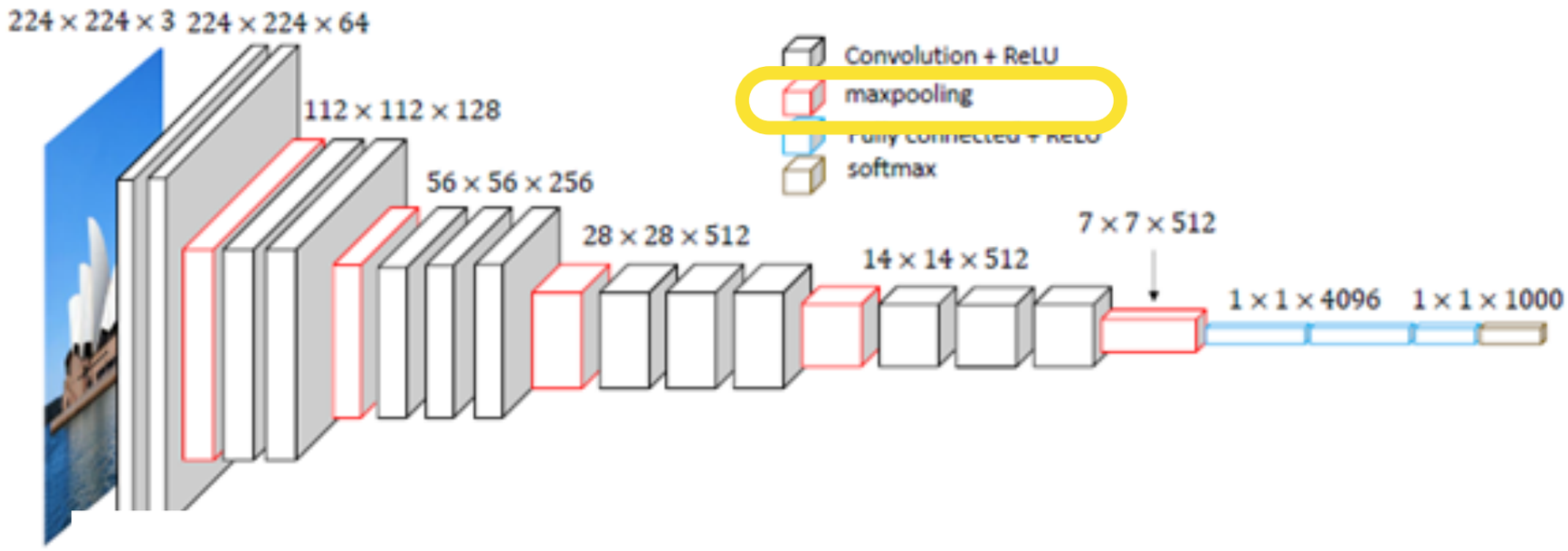
An example of max-pooling operator



Pooling downsamples the input layer



Basics of DL models



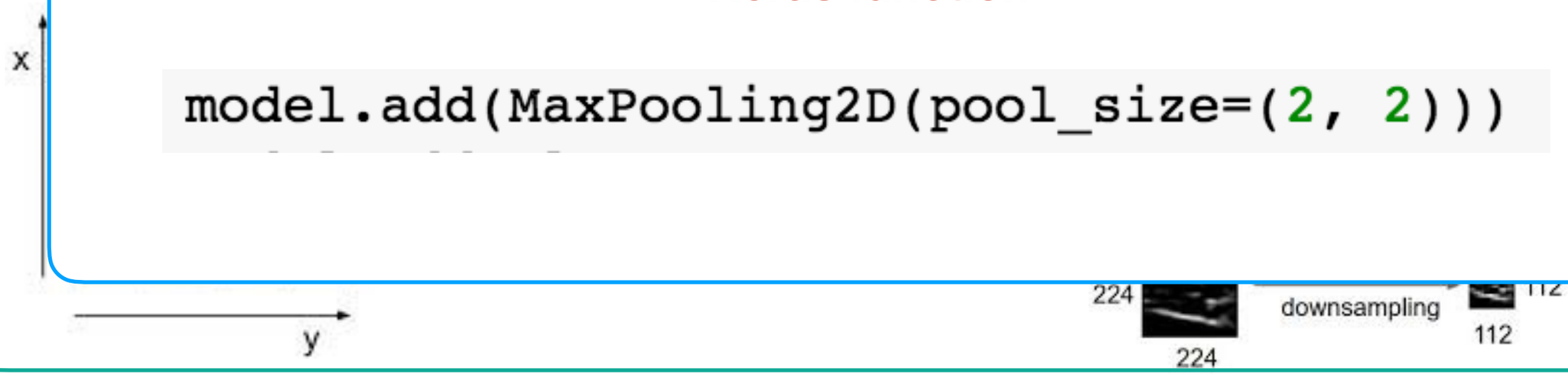
Elementary components	Convolution layers	Activation layers	Pooling layers	Dense layers

An example of max-pooling operator

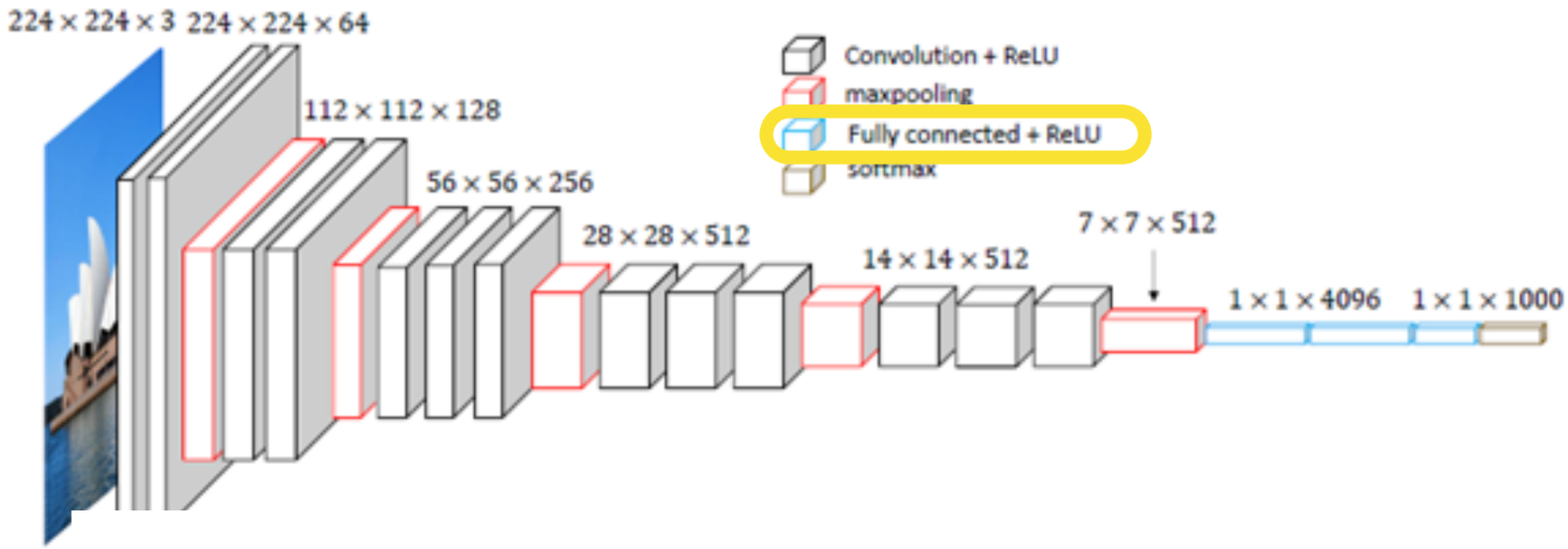
Pooling downsamples the input layer

Keras function

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```



Basics of DL models

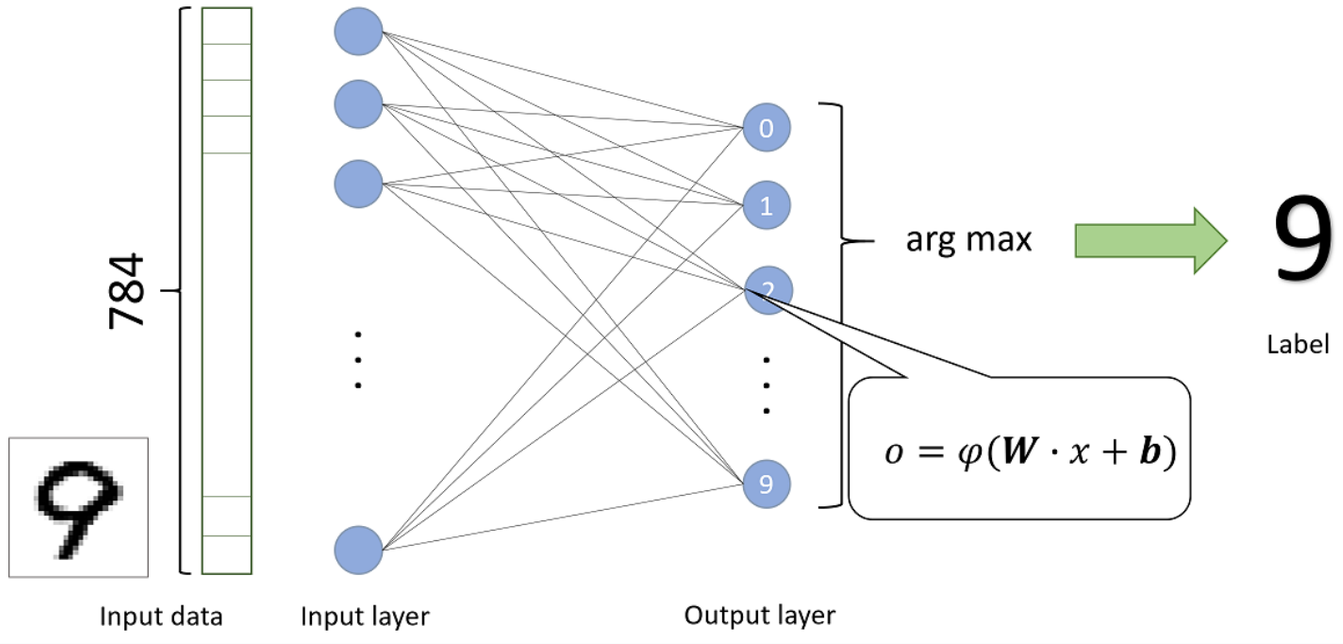


Elementary components

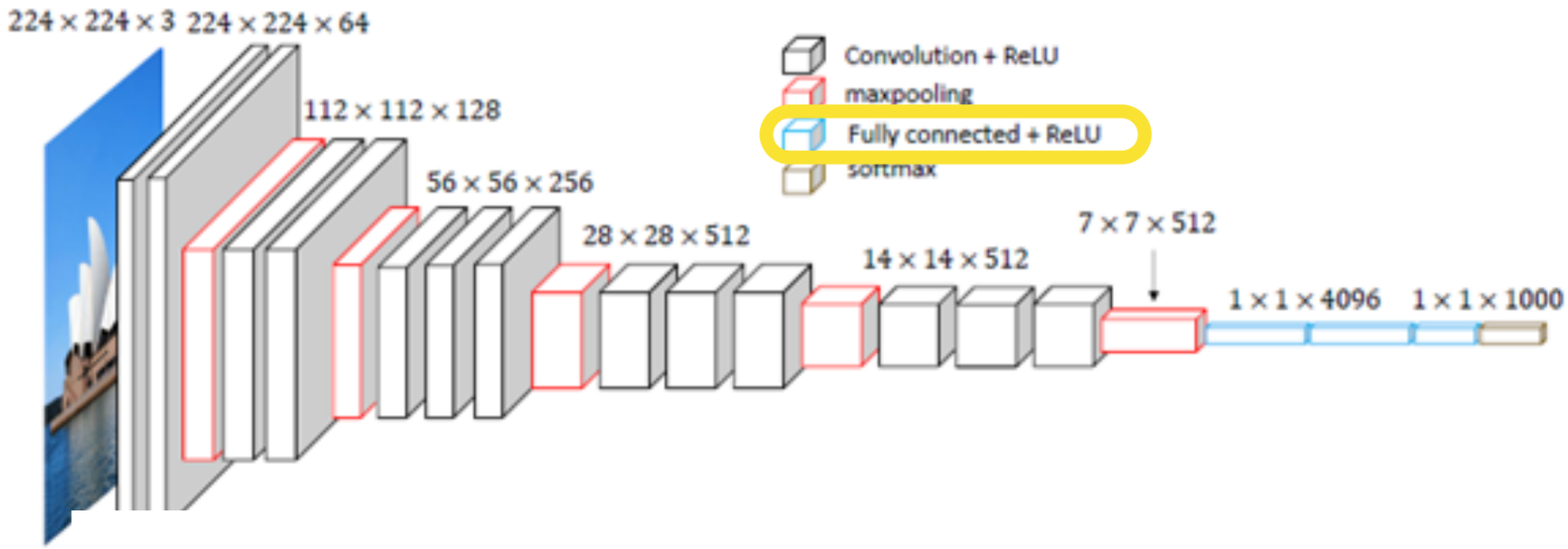
Convolution layers | Activation layers | Pooling layers | Dense layers

Dense layers
or
Fully-connected (FC) layer

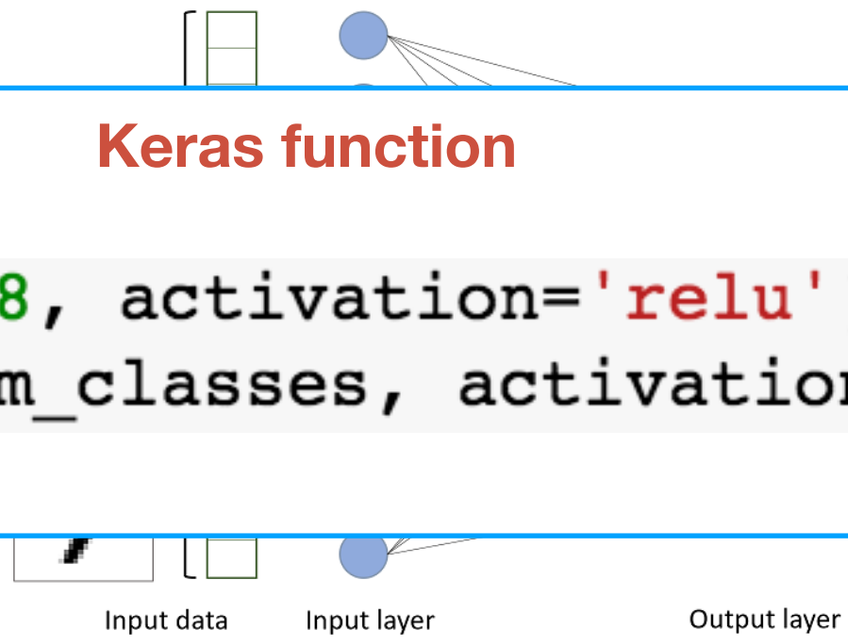
as in a classic MLP



Basics of DL models



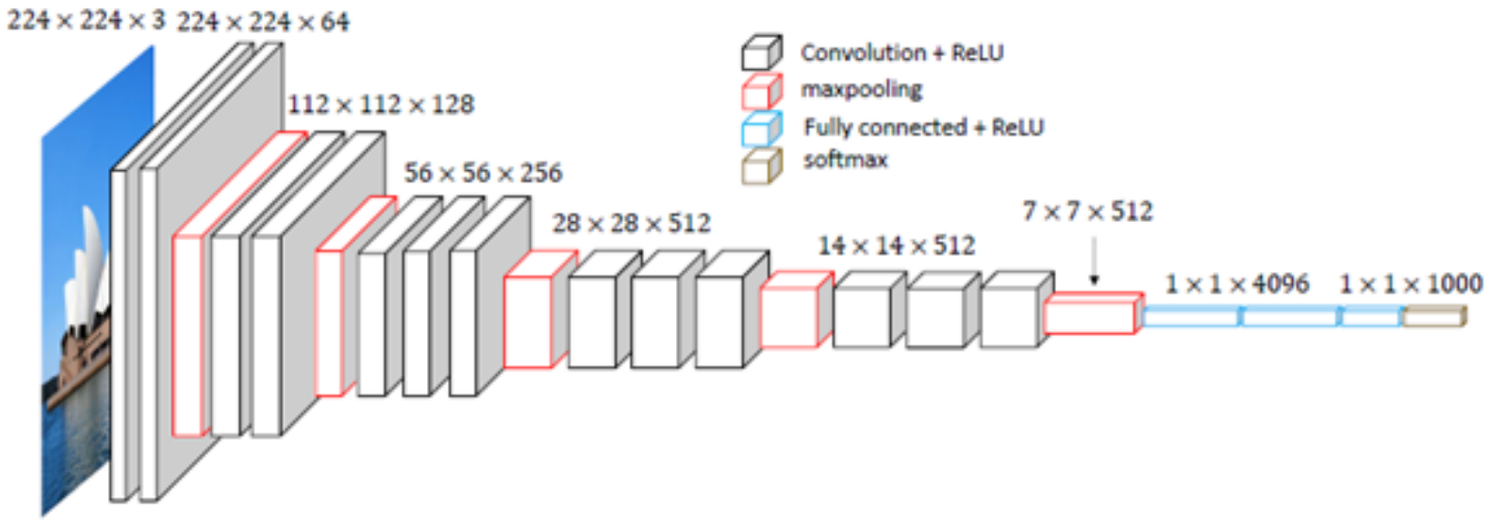
Elementary components	Convolution layers	Activation layers	Pooling layers	Dense layers



Keras function

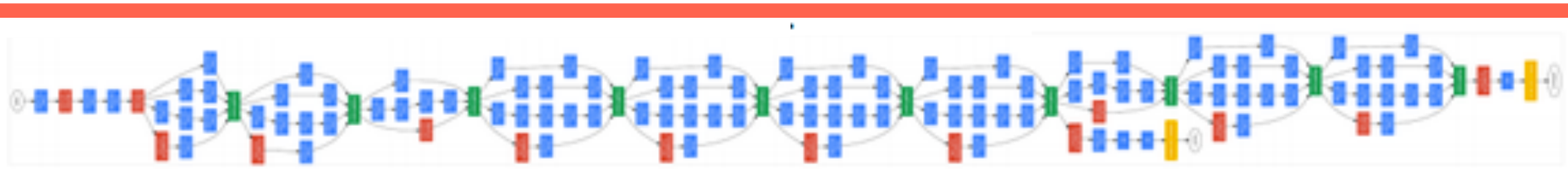
```
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Examples of DL models for object recognition

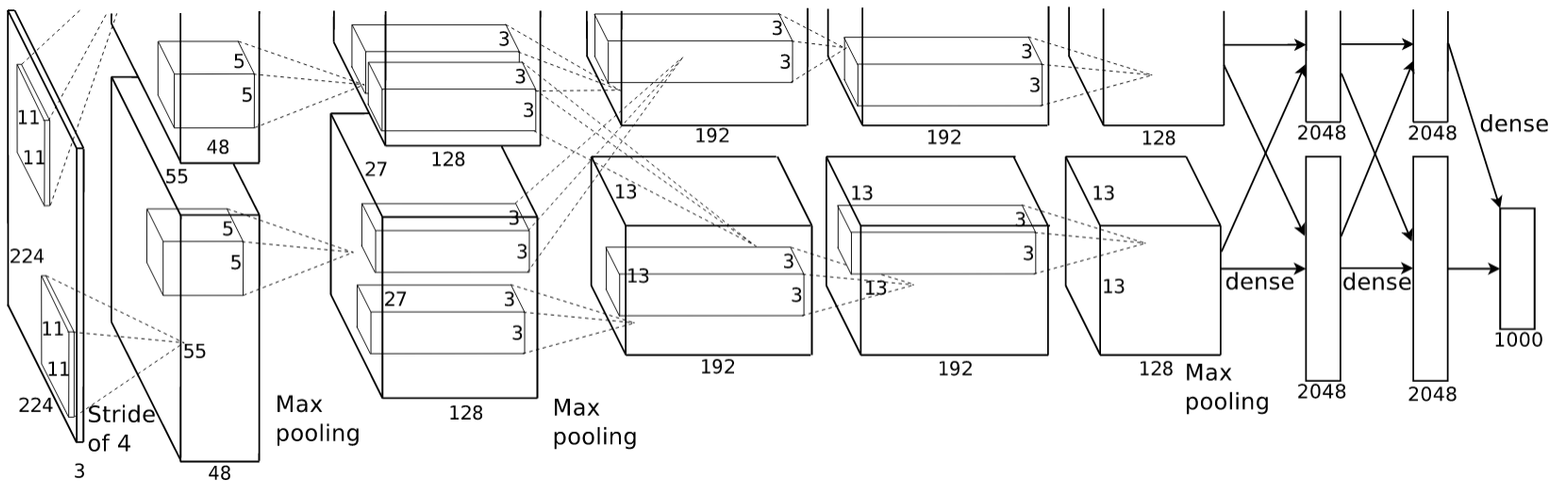


VGG16
(<100M of parameters)

Google` inception
(5M of parameters)



Convolution
Pooling
Softmax
Other

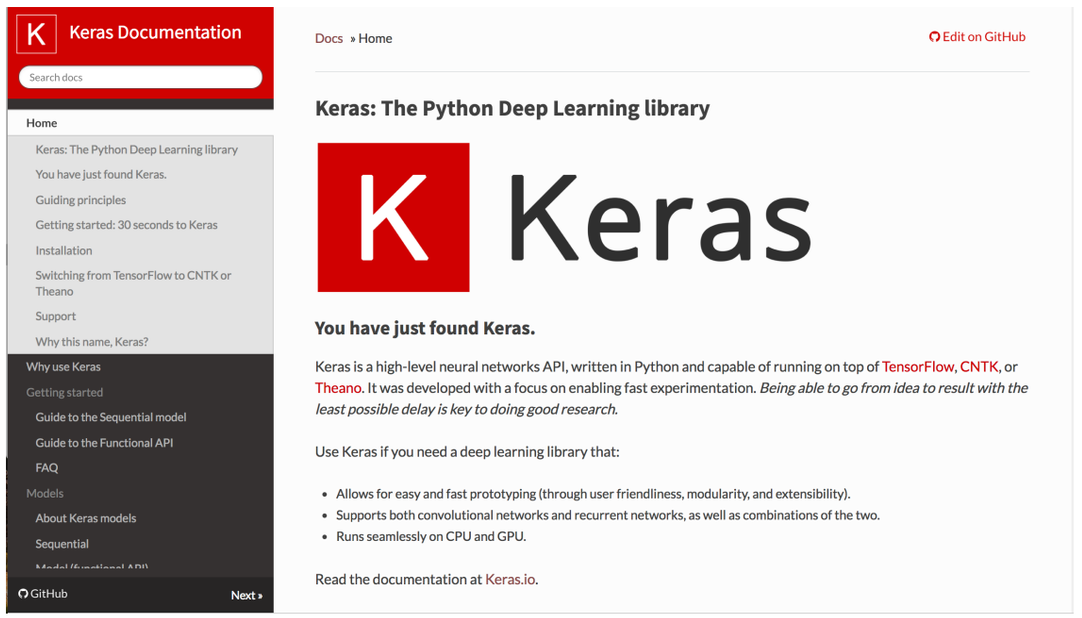


AlexNet
(60M of parameters)

How to implement Neural Nets with Keras ?

Using neural nets with Keras

- What is Keras ?
  - A high-level library for DL using Tensorflow or Theano as low-level DL library
- Not the only Python DL library



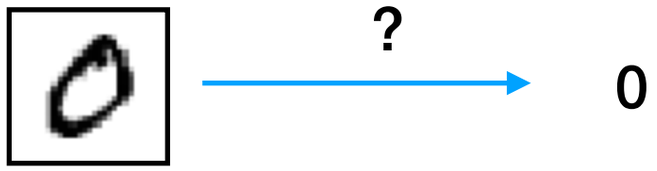
	PyTorch	Tensorflow	Keras
+	Mid-level, modelling flexibility, supported by Facebook AI Research	Low-level, computational efficiency, developed by Google	High-level, simple-to-use, benefits from using Tensorflow as backend
-	in-between in termes of use simplicity and computational complexity	relatively complex to develop with	Lower modelling flexibility

Load Python notebook

LabSession_MLP_MNIST.ipynb

Multilayer Perceptron (MLP)

- A simple regression example (MNIST case-study)
  - Goal: predict a digit from an image



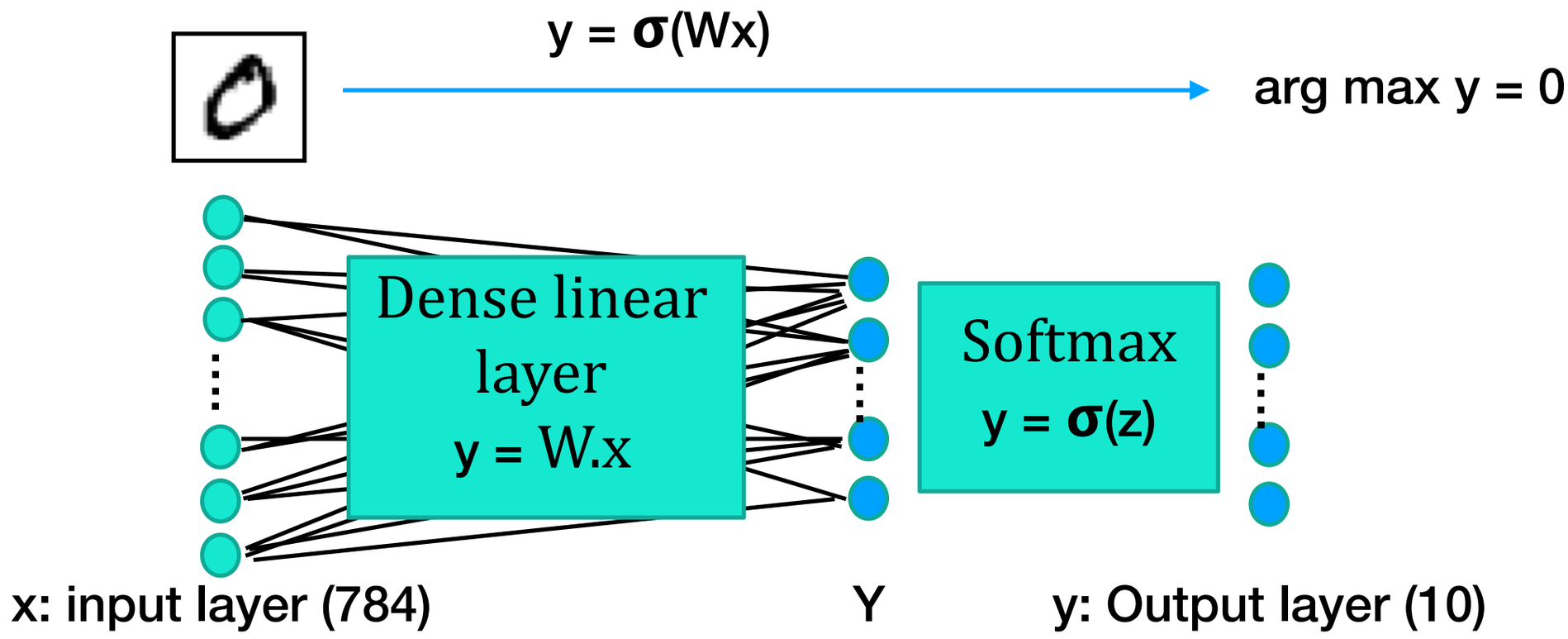
  - Formulation: non-linear regression using a softmax model

with $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 1, \ldots, K$.

- Softmax regression as a one-layer MLP ?

Using neural nets with Keras

- Back to Softmax regression as a one-layer MLP



Building a model with Keras

```
from keras.models import Sequential
from keras.layers import Dense

# y = softmax (Wx+b)
model = Sequential()
model.add(Dense(num_classes, activation='softmax', input_shape=(784,)))

model.summary()
```

Declare a new empty model

Add a dense layer

Using neural nets with Keras

- Back to Softmax regression as a one-layer MLP

Using a model with Keras

1. Model "compilation"

```
from keras.optimizers import RMSprop

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

training criterion

Learning algorithm

2. Model training from data

```
batch_size = 128
epochs = 20

history = model.fit(x_train, z_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, z_test))
```

training parameters (e.g., # of iterations)

training data

Test data

3. Application of a trained model to new data

```
z_pred = model.predict(x_test)
```

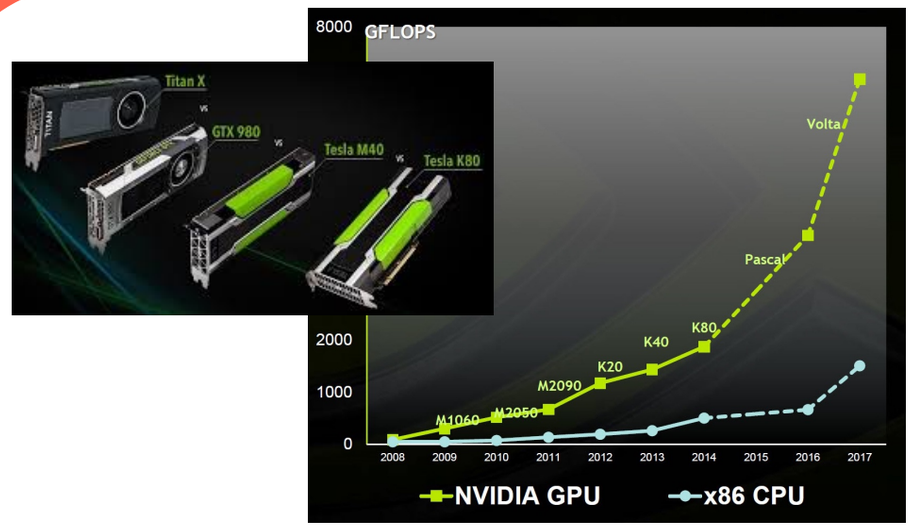
New input data

Load Python notebook

LabSession_CNN_MNIST.ipynb

Why are Deep Learning models so successful?

Key reasons for DL emergence



High-performance computing (GPU)

Large annotated dataset (> 1M)

+

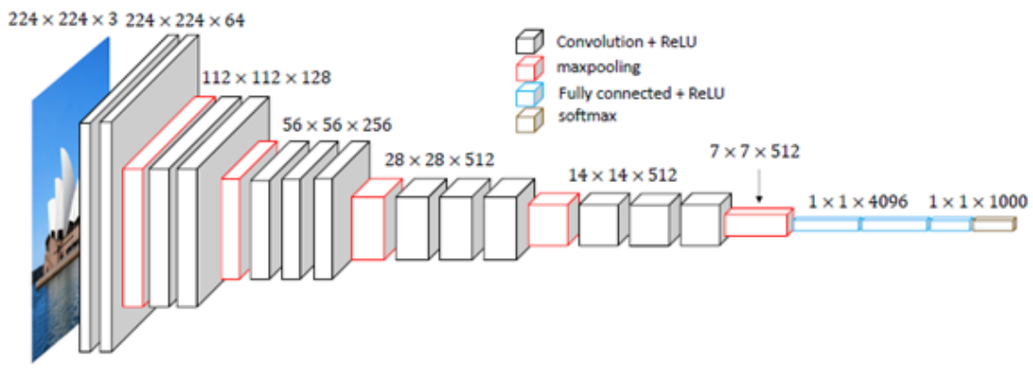
TensorFlow

Keras

Caffe

PYTORCH

Efficient & easy-to-use libraries

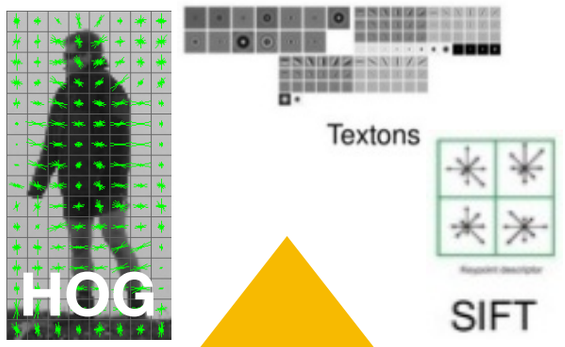


End-to-end learning

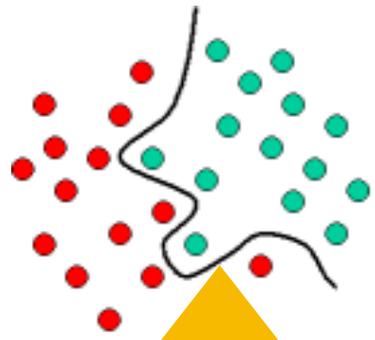
Shallow vs. Deep models

Shallow Learning

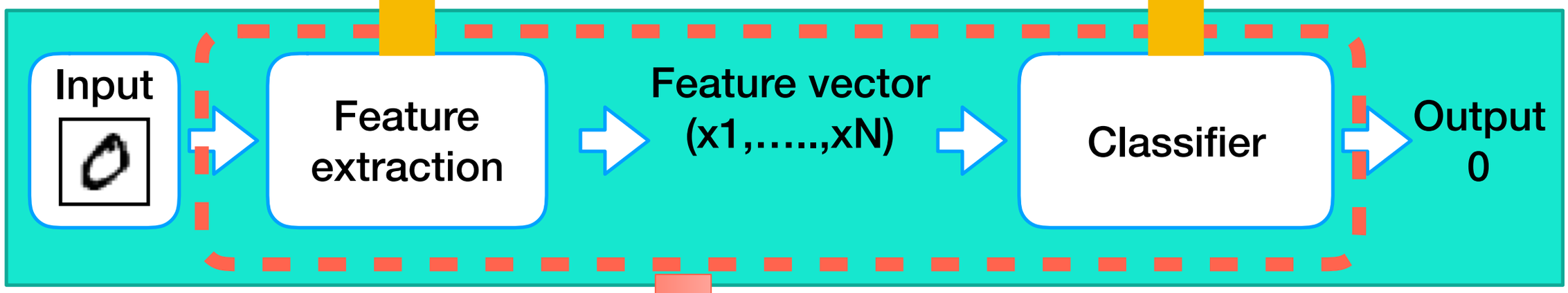
Handcrafted feature



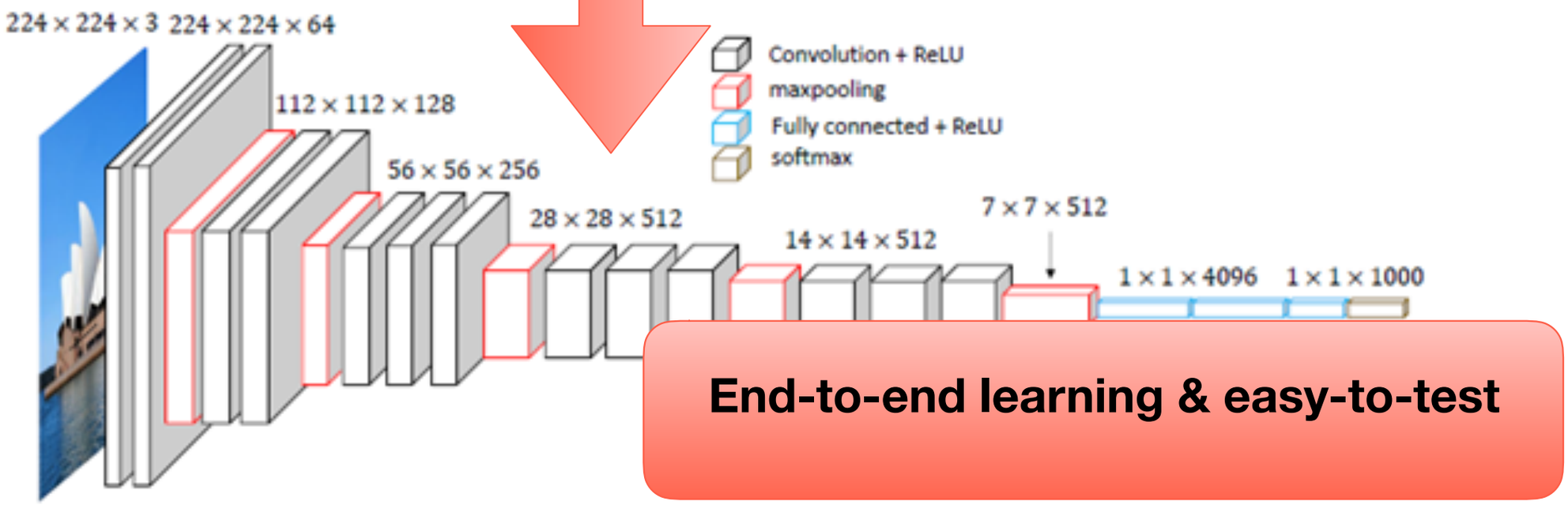
Classifier



Classic Workflow

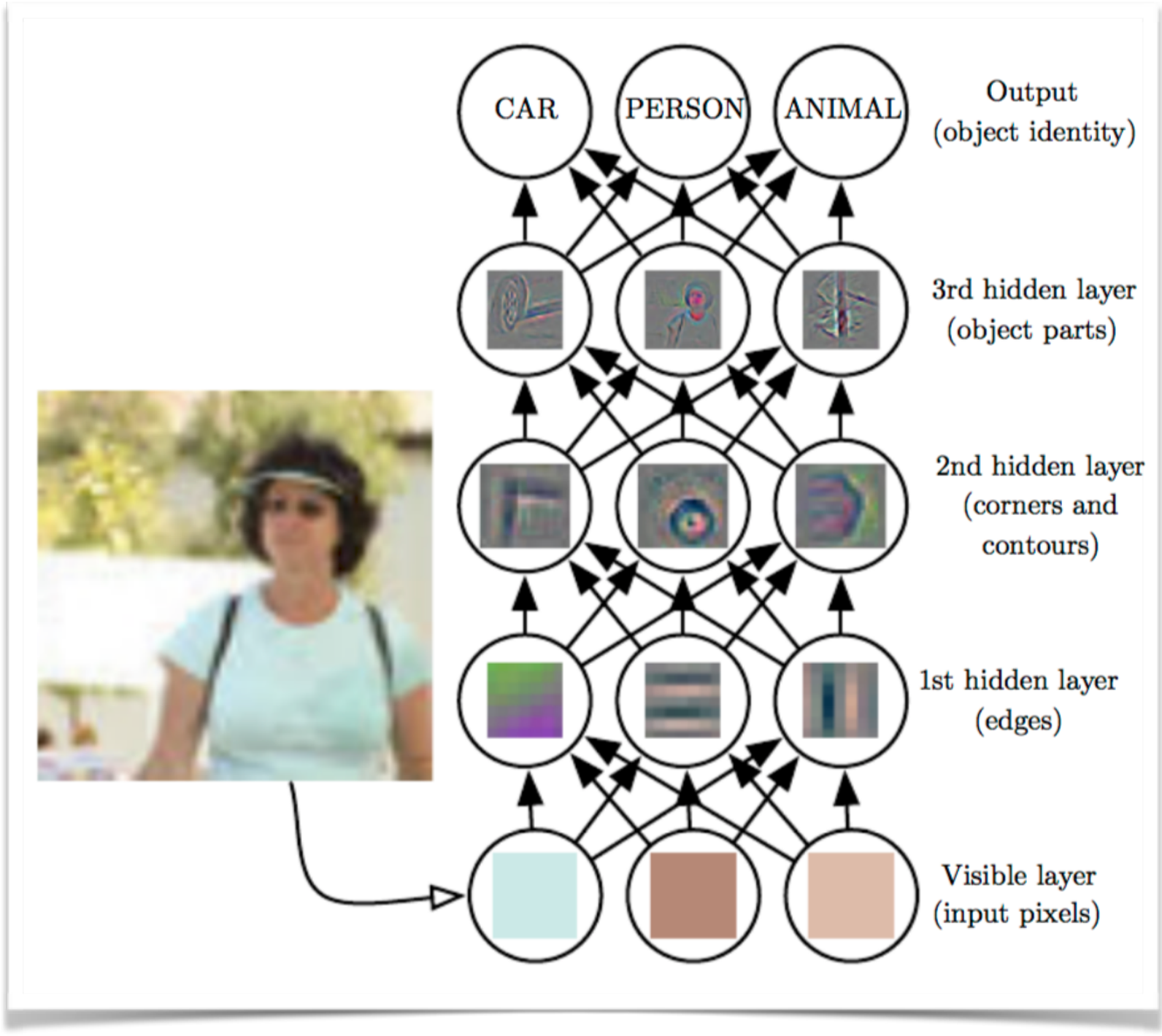


Deep Learning



End-to-end learning & easy-to-test

Feature extraction & Deep learning

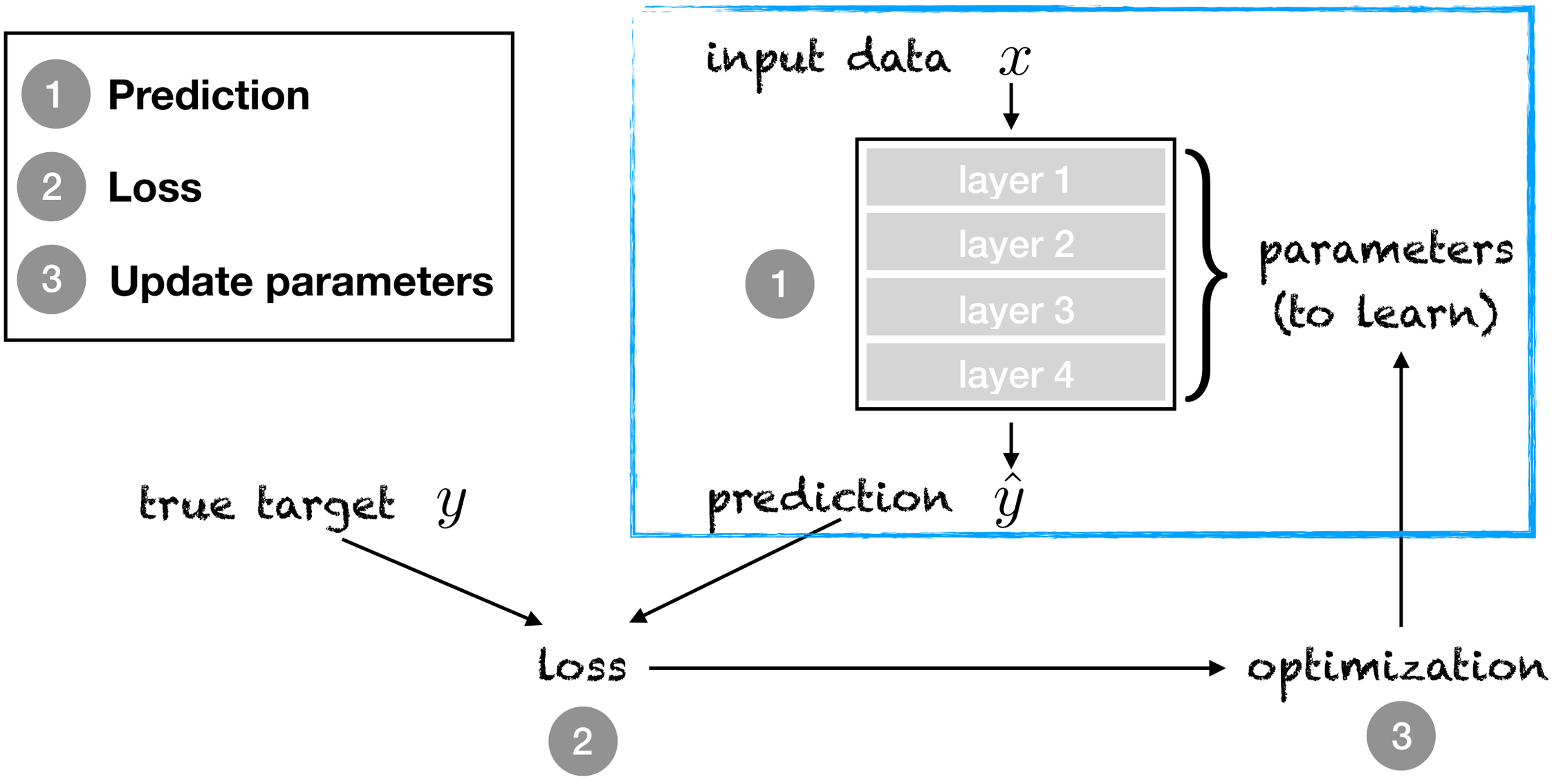


Summary

Tips for using DL models

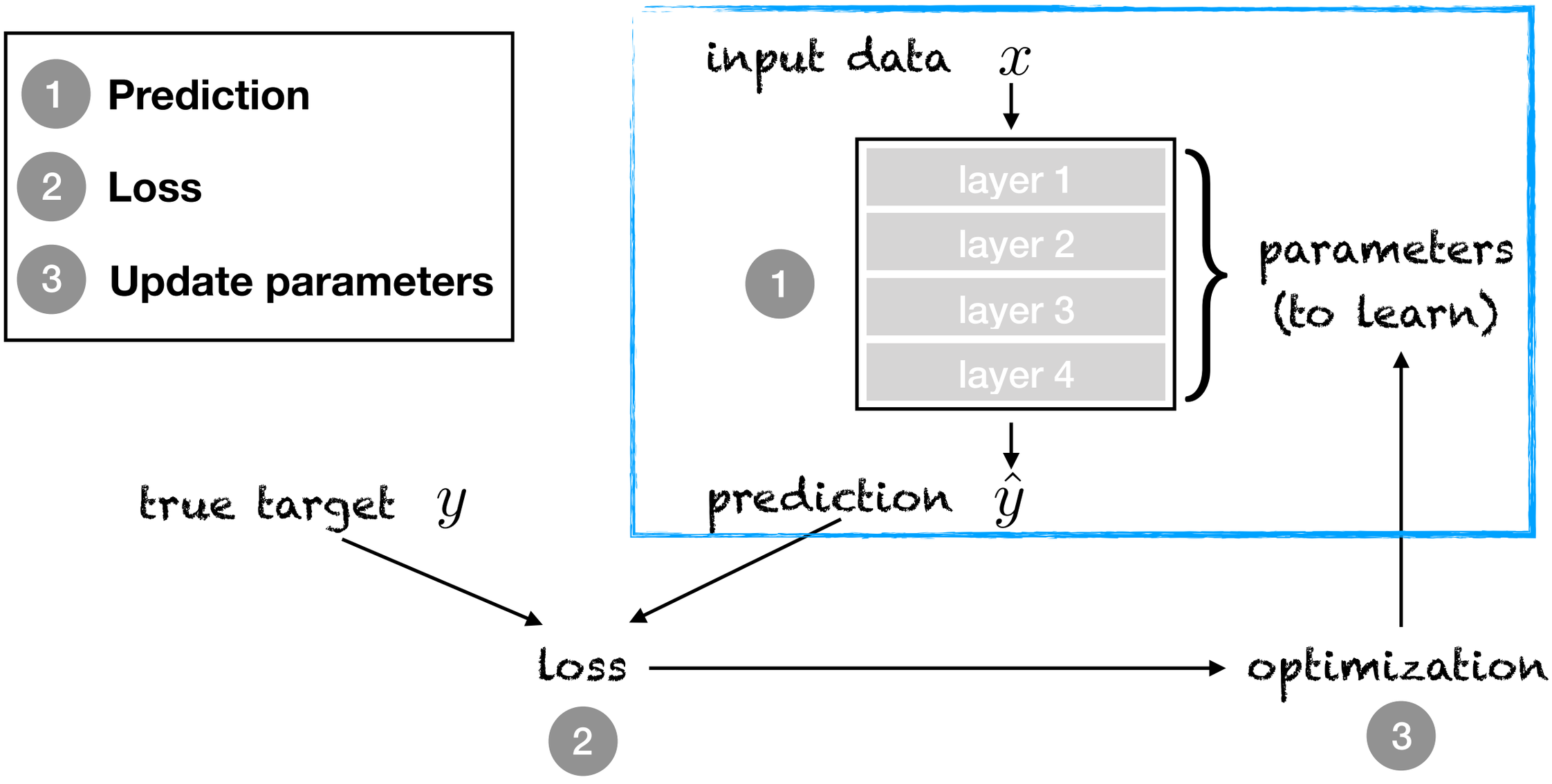
Overview

1. **Prediction**
2. **Loss**
3. **Update parameters**

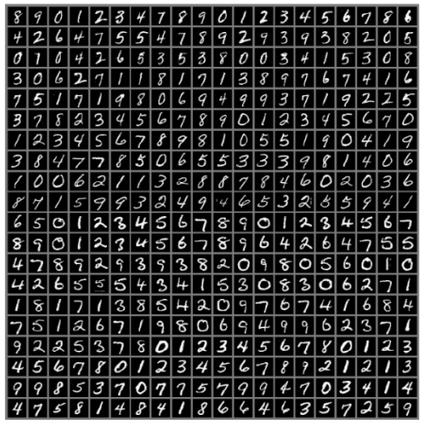


Overview

1. **Prediction**
2. **Loss**
3. **Update parameters**



Implementation vs theory



1. DATA

```
from keras.models import Sequential
from keras.layers import Dense

# y = softmax (Wx+b)
model = Sequential()
model.add(Dense(num_classes, activation='softmax', input_shape=(784,)))

model.summary()
```

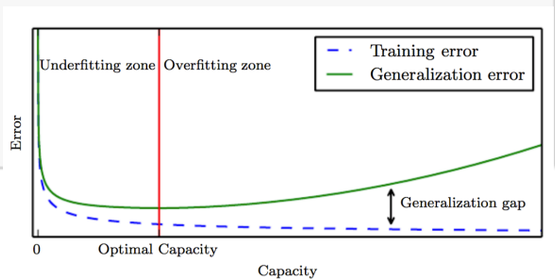
3. NETWORK

```
from keras.optimizers import RMSprop

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

2. LOSS FUNCTION

```
batch_size = 128
epochs = 20

history = model.fit(x_train, z_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, z_test))
```

4. LEARNING



Tips for using DL models

1. **State the considered issue as a machine learning problem**

2. **Build a (large-scale) groundtruthed dataset**

3. **Design a DL architecture (combination of conv, pooling, dense layers)**
   Possibility for using existing architectures (fine-tuning, transfer learning)

4. **Buy or rent computational ressources (especially, GPU resources)**

5. **Train the selected DL architecture**
   Here comes the “tricky” part of the selection of the optimisation strategy

Training and Optimisation issues in Deep Learning

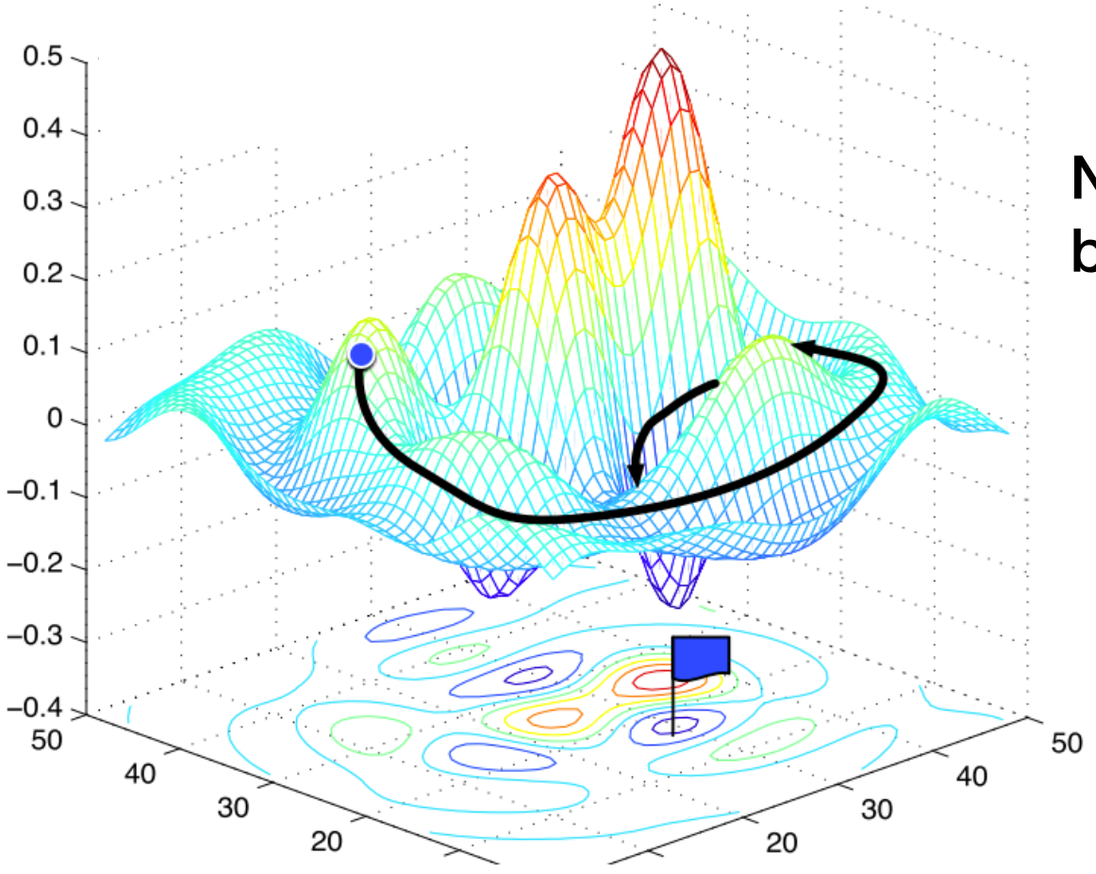
All you need is ..... Data, GPUs

And a good optimisation algorithm

Training and Optimisation issues in Deep Learning

All you need is ..... Data, GPUs

And a good optimisation algorithm



Non-convex optimisation using gradient-based algorithm

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

Model parameters

Gradient of the training loss

A short tour of the NNs zoo

2 (at least) key aspects

NN architecture (parameterization of model f)
Training loss

Autoencoder and dimensionaly reduction

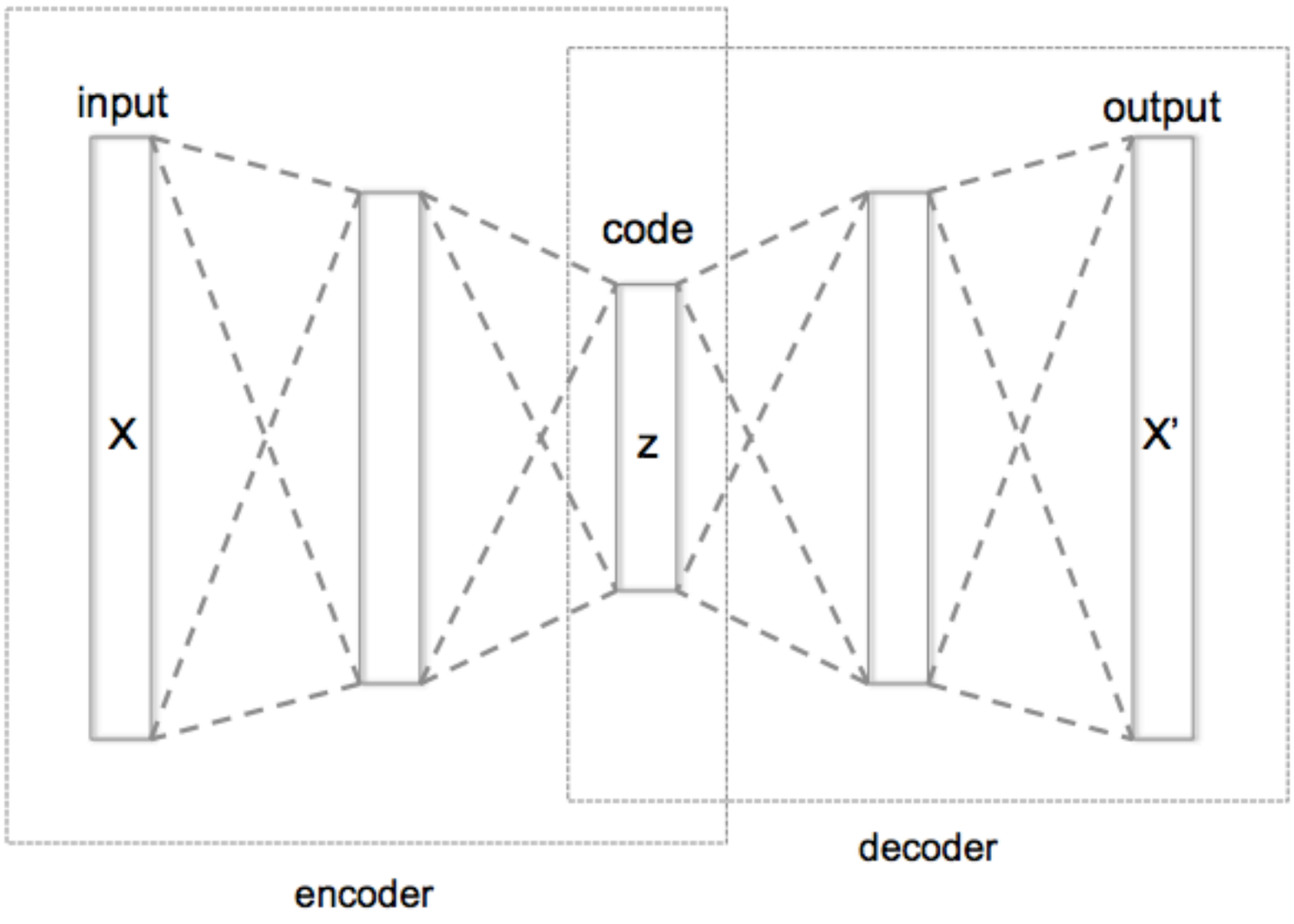
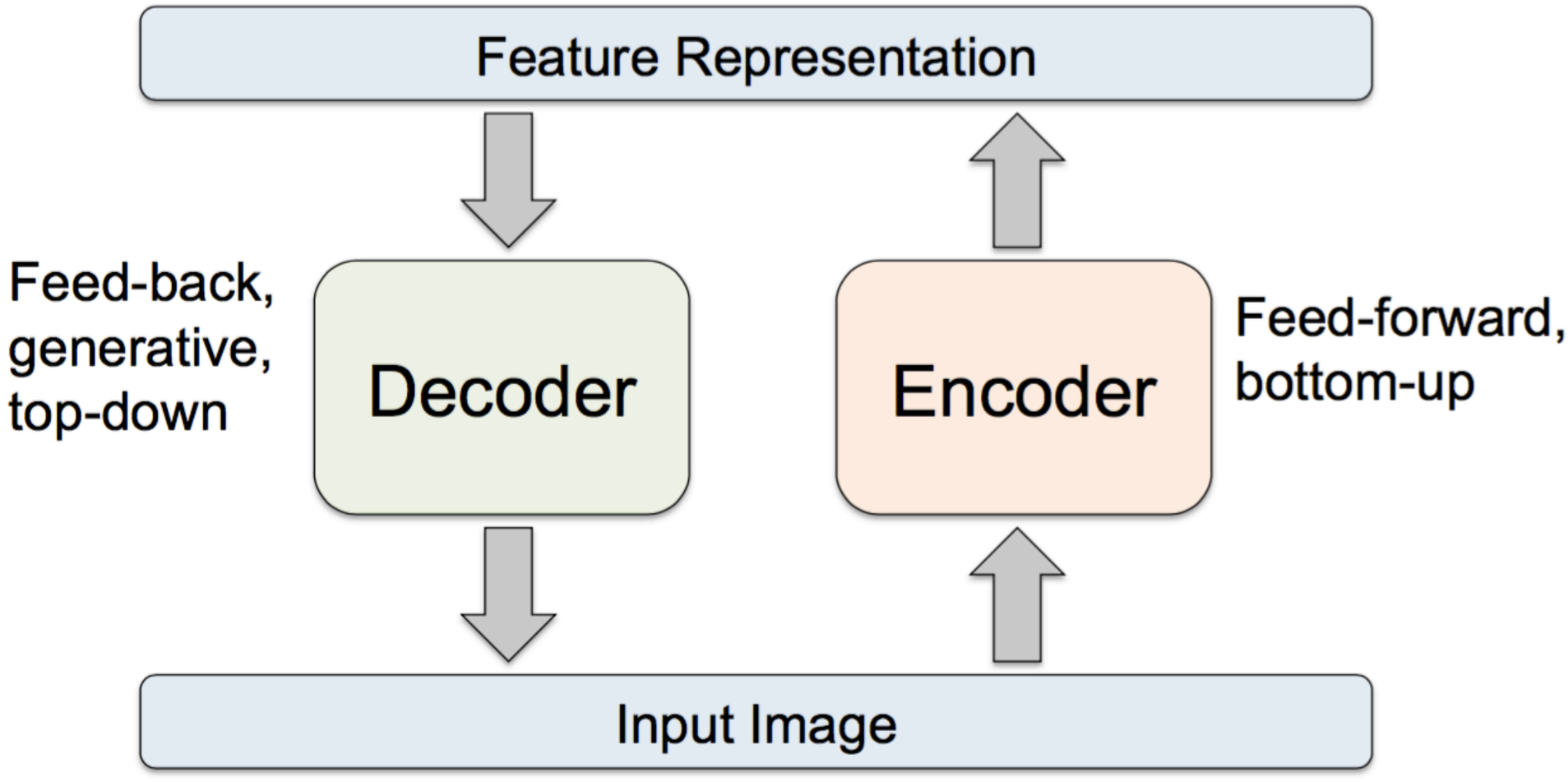


Figure from wikipedia

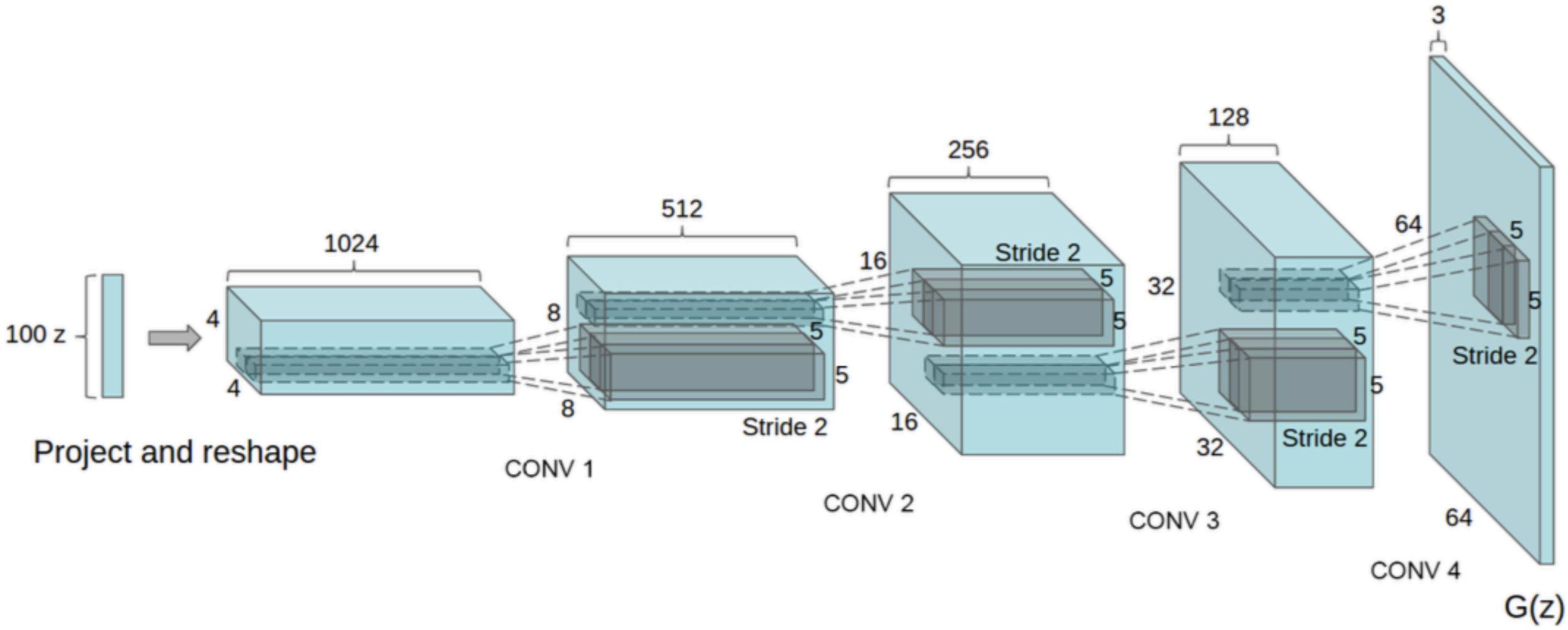
Autoencoders

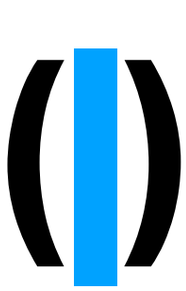


- Details of what goes insider the encoder and decoder matter!
- Need constraints to avoid learning an identity.

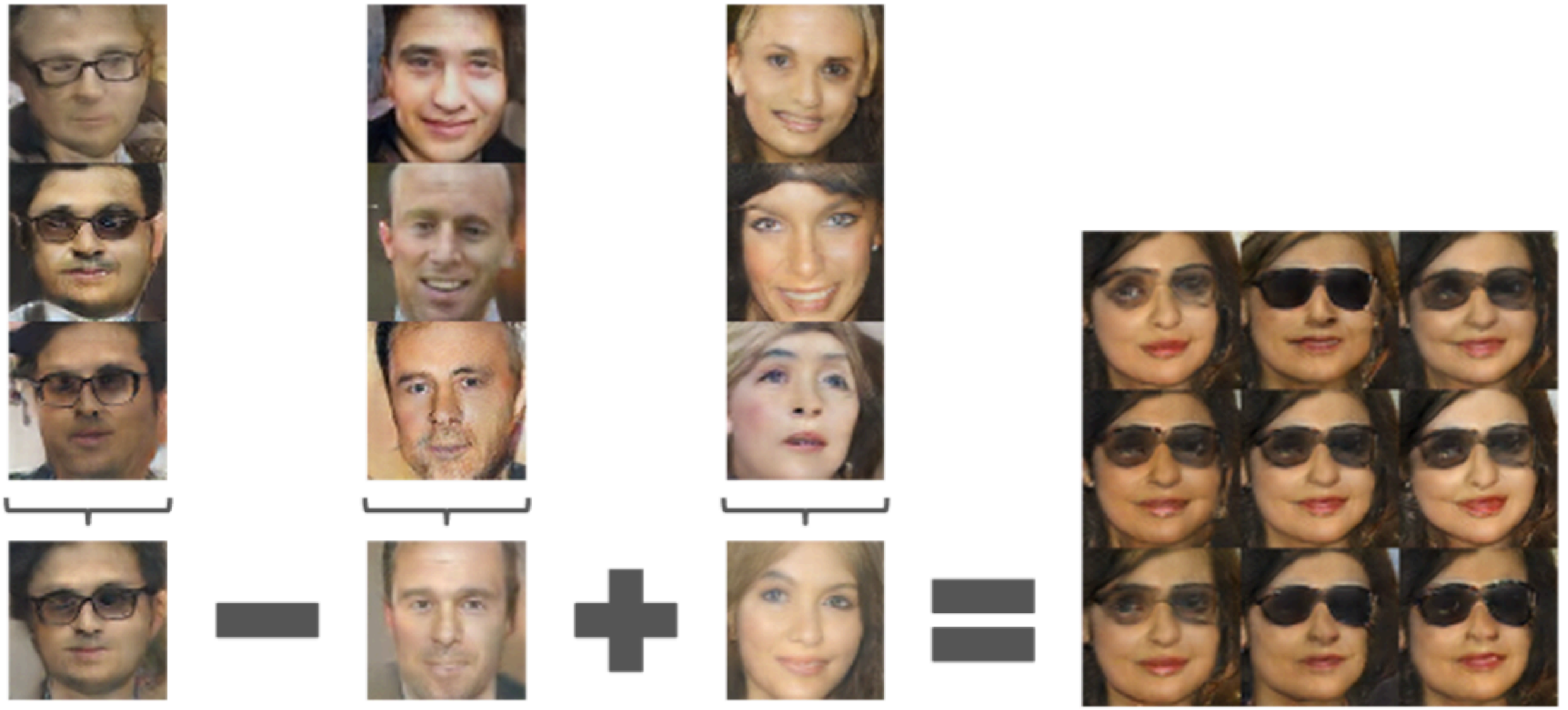
Slide from Salakhutdinov, KDD, 2014

Generative Models (e.g., GAN)





Generative Modeling

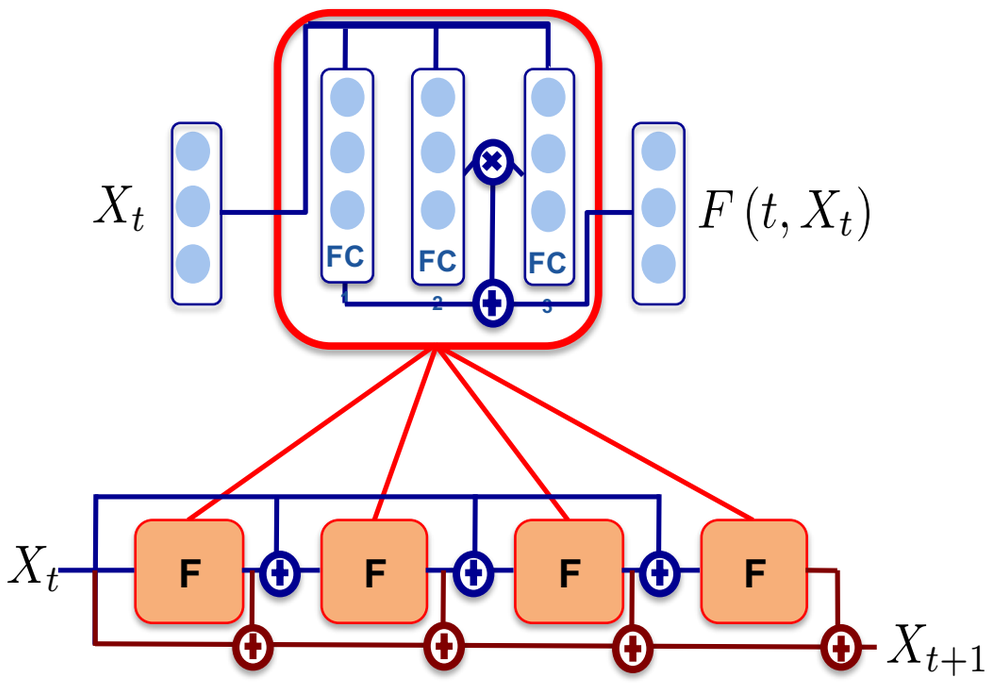


Radford et al. 2015

NNs for ODE

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \sigma\left(y(t) - x(t)\right)$$
$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = x(t)\left(\rho - z(t)\right) - y(t)$$
$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = x(t)\,y(t) - \beta\,z(t)$$

Lorenz-63 equations



Two key steps:

- **NN parameterisation for operator F**

$$d_t X_t = F_\theta(X_t)$$

- **NN parameterization for the associated integration scheme**

Adjoint operator:

- **Adjoint of the forward integration scheme**
- **Backward NN integration of Adjoint of the forward integration scheme**

Losses & training strategies

Synoptic (point-wise) losses

$$\arg\min_{\theta} \sum_{i} \|y_i - f\theta(x_i)\|_p^p$$

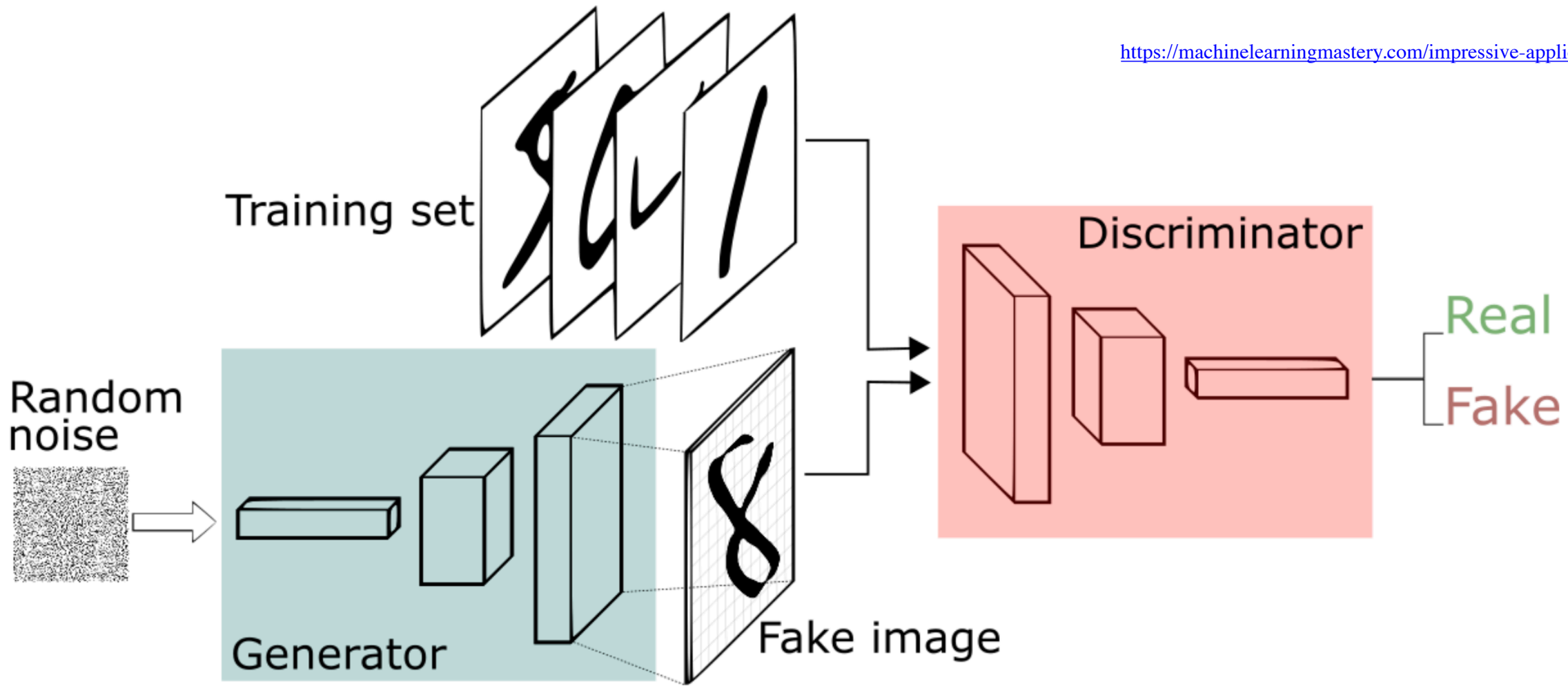
Likelihood losses

$$\arg\min_{\theta} \sum_{i} \left(y_i - f_\theta(x_i)\right)^t \Sigma_\theta^{-1} \left(y_i - f_\theta(x_i)\right) - 1/n \log|\Sigma_\theta|$$

Distribution-based cost (Optimal transport)

Generative Adversarial Networks

How to deal with situations where one cannot define an explicit criterion to train a generative model but numerous examples are available ?



https://machinelearningmastery.com/impressive-applications-of-

Image credit: Thalles Silva

Generative Adversarial Networks

Image processing examples

https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/